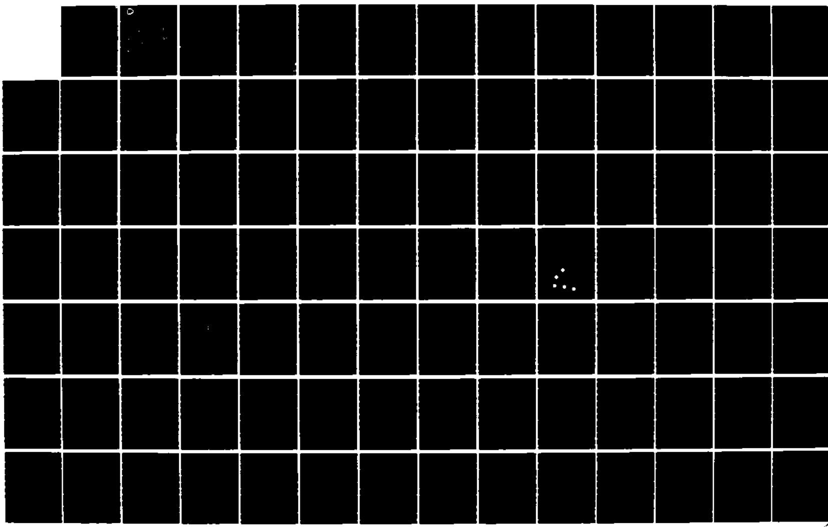
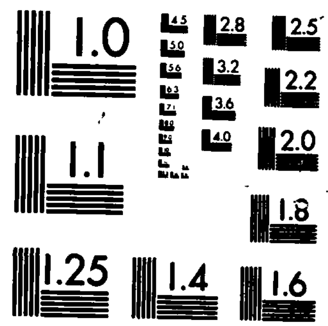
AD-A165 123 ADA (TRADEMARK) TRAINING CURRICULUM: SOFTWARE ENGINEERING FOR MANAGERS M101 TEACHER'S GUIDE(U) SOFTECH INC WALTHAM MA 1986 DAAB07-83-C-K506 1/4

UNCLASSIFIED F/G 5/9 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

AD-A165 123 1986

Ada® Training Curriculum

Software Engineering For Managers
M101
Teacher's Guide

86 3 11 147

Supersedes AD-A142432

U.S. Army Communications-Electronics Command
(CECOM)

Contract DAAB07-83-C-K506

Prepared By:

SOFTECH, INC.
460 Totten Pond Road
Waltham, MA 02154

SOFTWARE ENGINEERING FOR MANAGERS

M101

Dist	Special
A-1	

QUALITY INSPECTED 3

VG 742.1

INSTRUCTOR NOTES

Contents:

I. 1) BACKGROUND AND MOTIVATION –

- DEFINITION OF SOFTWARE ENGINEERING, and
- MOTIVATION FOR SOFTWARE ENGINEERING (SOFTWARE CRISIS);

II. 2) SOFTWARE ENGINEERING AND ITS GOALS –

- SOFTWARE ENGINEERING GOALS, and
- SOFTWARE ENGINEERING PRINCIPLES;

III. 3) ACHIEVING SOFTWARE ENGINEERING GOALS –

- SOFTWARE LIFE-CYCLE,
- METHODS AND TOOLS FOR EACH PHASE OF LIFE-CYCLE,
- TESTING, and
- SOFTWARE MANAGEMENT;

IV. 4) SOFTWARE ENGINEERING AND Ada –

- RELATIONSHIP OF SOFTWARE ENGINEERING TO Ada.

TOPICS TO BE COVERED:

BACKGROUND AND MOTIVATION

SOFTWARE ENGINEERING AND ITS GOALS

ACHIEVING SOFTWARE ENGINEERING GOALS

SOFTWARE ENGINEERING AND Ada

INSTRUCTOR NOTES

<u>THEME</u>: SOFTWARE ENGINEERING IS AN APPROACH (OR SET OF APPROACHES) TO ADDRESSING THE "SOFTWARE CRISIS"

<u>PURPOSE</u>: TO MOTIVATE THE STUDY OF SOFTWARE ENGINEERING AND METHODOLOGIES

<u>REFERENCES</u>: J.D. MUSA (EDITOR), "STIMULATING SOFTWARE ENGINEERING PROGRESS - A REPORT OF THE SOFTWARE ENGINEERING PLANNING GROUP", ACM SIGSOFT SOFTWARE ENGINEERING NOTES, VOL 8, NO. 2, APRIL 1983

PART I

BACKGROUND AND MOTIVATION

INSTRUCTOR NOTES

SECTION 1

DEFINITION OF SOFTWARE ENGINEERING

INSTRUCTOR NOTES

NO COMMON, SIMPLE DEFINITION FOR SOFTWARE ENGINEERING.

SEEWG = SOFTWARE ENGINEERING ENVIRONMENT WORKING GROUP, WORKING GROUP TO DEFINE REQUIREMENTS FOR A NAVY STANDARD SOFTWARE ENGINEERING ENVIRONMENT.

SOME DEFINITIONS

- "SOFTWARE ENGINEERING IS THE APPLICATION OF SCIENCE AND MATHEMATICS BY WHICH THE CAPABILITIES OF COMPUTER EQUIPMENT ARE MADE USEFUL TO MAN VIA COMPUTER PROGRAMS, PROCEDURES AND ASSOCIATED DOCUMENTATION."

  BEOHM 1981

- "... A SOFTWARE ENGINEERING PROCESS IS A SET OF ACTIVITIES FOR DEVELOPING AND MODIFYING SOFTWARE THROUGH ITS LIFE CYCLE."

  SEEWG REPORT 1982

- "A METHODOLOGY IS A REPEATABLE HUMAN PROCEDURE WHICH SUPPORTS SOME ASPECT OF AN ACTIVITY."

  SEEWG REPORT 1982

INSTRUCTOR NOTES

SECTION 2

MOTIVATION FOR SOFTWARE ENGINEERING

INSTRUCTOR NOTES

THIS IS THE DEFINITION OF THE SOFTWARE CRISIS THAT MOTIVATED THE DEVELOPMENT OF ADA.

GIVE SOME PERSONAL EXAMPLES OF SYSTEMS YOU HAVE WORKED ON OR ASK THE CLASS FOR SOME.

MOTIVATION FOR SOFTWARE ENGINEERING

(SOFTWARE CRISIS)

- SOFTWARE FOR COMPLEX MILITARY SYSTEMS
  - IS USUALLY LATE
  - COSTS MORE THAN ORIGINALLY ESTIMATED
  - DOES NOT WORK TO ORIGINAL SPECIFICATIONS
  - IS UNRELIABLE
  - IS DIFFICULT AND COSTLY TO MAINTAIN

INSTRUCTOR NOTES

TALK TO EACH BULLET, GIVING THE CLASS PERSONAL EXPERIENCE OR TRY TO GET THEM TO RELATE THEIR EXPERIENCES.

NOTE: DURING THIS SECTION YOU ARE TRYING TO SET THE STAGE FOR LATER PARTICIPATION BY THE CLASS. MAKE THEM FEEL AS IF THEY "OWN" THESE PROBLEMS.

MOTIVATION FOR SOFTWARE ENGINEERING

(ADDITIONAL PROBLEMS)

- SOFTWARE IS NOT REUSABLE ON DIFFERENT SYSTEMS
- PROLIFERATION OF METHODS, LANGUAGES AND ARCHITECTURES
- METHODS AND LANGUAGES NOT SUITED FOR CURRENT APPLICATIONS
- SUPPLY OF QUALITY SOFTWARE PERSONNEL NOT ABLE TO MEET CURRENT SOFTWARE DEMAND
- SOFTWARE TASKS ARE MORE COMPLEX NOW, BUT NO WIDELY USED METHODS AND TOOLS TO DEAL WITH THE PROBLEM EXIST
- LACK OF ADEQUATE MANAGEMENT <u>AND</u> SOFTWARE DEVELOPMENT METHODS/TOOLS

INSTRUCTOR NOTES

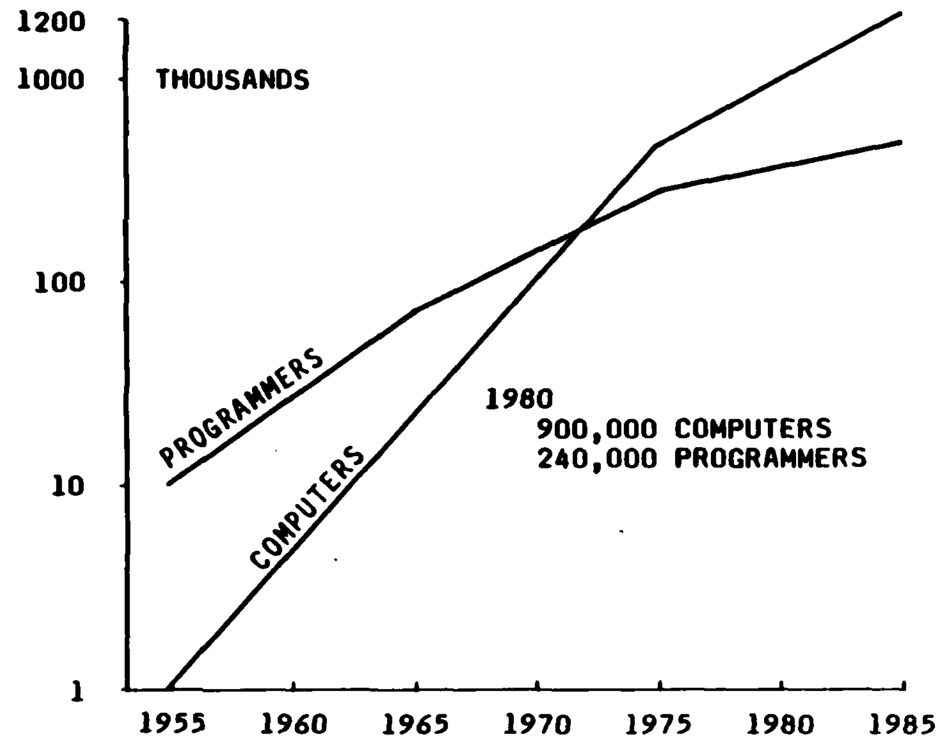
THE KEY POINT IS TO NOTE THE RATE OF GROWTH.

ENVIRONMENT FACING SOFTWARE ENGINEERING

- DEMAND FOR SOFTWARE OUTSTRIPPING DEVELOPMENT CAPABILITY

TOTAL U.S. DATA PROCESSING INDUSTRY EXPENDITURES

Year	Expenditure (billions of 1970 dollars)	Percent of GNP
1970	21	2.1
1975	41	3.2
1980	82	5.2
1985	164	8.3



- PRODUCTIVITY ADVANCES ARE NOT KEEPING UP

SOURCE: REPORT OF THE SOFTWARE ENGINEERING GROUP, 1983

VG 742.1

INSTRUCTOR NOTES

PRODUCTIVITY IS A MEASURE OF HOW EFFICIENTLY WE CAN PRODUCE SOFTWARE (NOTE THE SLOW RATE OF IMPROVEMENT).

PRODUCTION IS A MEASURE OF HOW MUCH THE DEMAND FOR SOFTWARE HAS INCREASED.

RELATIVE PROGRAMMER PRODUCTIVITY AND TOTAL U.S. YEARLY

CODE PRODUCTION (NORMALIZED TO 1955)

YEAR	PRODUCTIVITY	PRODUCTION
1955	1	1
1960	1.6	5
1965	2.0	16
1970	2.3	38
1975	2.7	59
1980	3.1	85
1985	3.6	119

- WE HAVE A SERIOUS DEFICIENCY
  - NEED MORE SOFTWARE DEVELOPERS
  - MUST MAKE EXISTING ONES MORE PRODUCTIVE

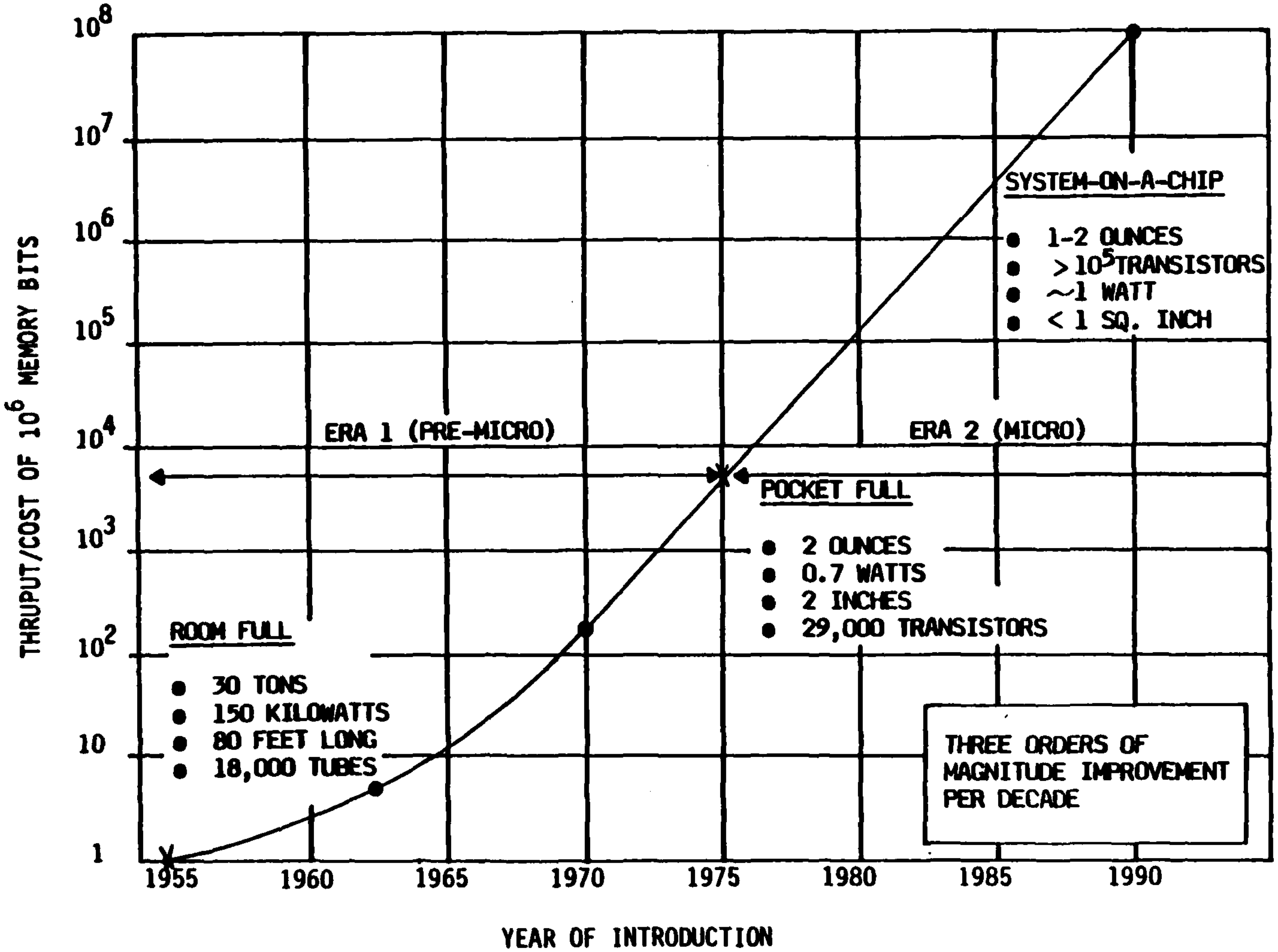
INSTRUCTOR NOTES

POINT OUT THAT HARDWARE TECHNOLOGY IS GROWING SO FAST THAT WE DON'T KNOW HOW TO PROPERLY UTILIZE IT.

NOTE THE Y AXIS IS A WAY OF MEASURING THE COMBINED EFFECT OF FASTER HARDWARE TECHNOLOGIES AND HIGHER DENSITY MICRO-CIRCUIT TECHNOLOGIES.

COMPUTER HARDWARE TECHNOLOGY

RELATIVE COST EFFECTIVENESS

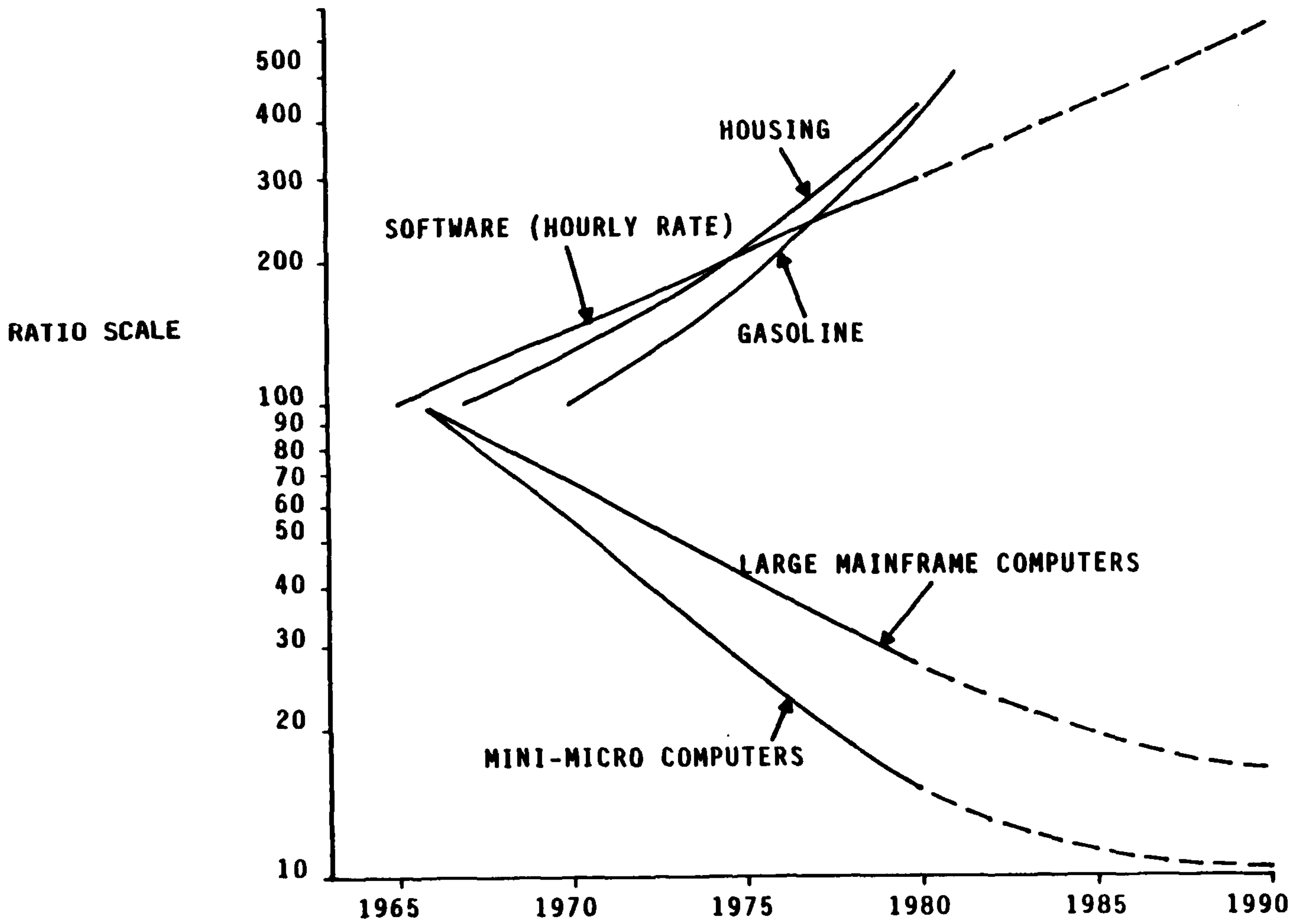


SOURCE: REPORT OF THE SOFTWARE ENGINEERING GROUP, 1983

INSTRUCTOR NOTES

NOTE THE DIRECTION OF THE VARIOUS COST FACTORS NOT THE ACTUAL VALUES.

SOFTWARE VS. HARDWARE COST TRENDS



SOURCE: REPORT OF THE SOFTWARE ENGINEERING GROUP, 1983

VG 742.1

II. S.E. GOALS AND PRINCIPLES.

THIS PART SHARPENS OUR FOCUS ON SOFTWARE ENGINEERING BY JUST DISCUSSING OBJECTIVES AND GOALS AND THEN BY INTRODUCING THE PRINCIPLES OF SOFTWARE ENGINEERING THAT ALLOW US TO ACHIEVE THESE GOALS.

IN SECTION 3 WE WILL REVIEW VARIOUS METHODS AND TOOLS FOR ACHIEVING THESE GOALS. WHEN WE DISCUSS THESE METHODS AND TOOLS WE WILL IDENTIFY THE PRINCIPLES THAT THEY ARE BASED ON.

ALLOW ABOUT 45 MINUTES TO 1 HOUR FOR THIS PART.

PART II

SOFTWARE ENGINEERING GOALS AND PRINCIPLES

INSTRUCTOR NOTES

SECTION 3

SOFTWARE ENGINEERING GOALS

INSTRUCTOR NOTES

THERE EXISTS AN ALMOST LIMITLESS LIST OF POTENTIAL GOALS FOR SOFTWARE ENGINEERING TO ATTAIN. THESE GOALS ARE NOT INDEPENDENT AND NOT ALL ARE APPROPRIATE IN ALL SITUATIONS. WE NEED TO IDENTIFY THE DEPENDENCIES AND RELATIONSHIPS THAT EXIST, UNDERSTAND HOW OVERALL SYSTEM OBJECTIVES INFLUENCE THEIR IMPORTANCE, AND ALSO UNDERSTAND THAT SOME GOALS ARE MUTUALLY EXCLUSIVE OR AT LEAST EXHIBIT TENDENCIES IN THAT DIRECTION.

POTENTIAL SOFTWARE ENGINEERING GOALS

CORRECTNESS	USABILITY	FLEXIBILITY	INTEROPERABILITY
ACCEPTABILITY	OPERABILITY	ADAPTABILITY	COMPLEXITY
COMPLETENESS	HUMAN FACTORS	EXTENSIBILITY	MODULARITY
CONSISTENCY	COMMUNICATIVENESS	ACCESSIBILITY	STRUCTUREDNESS
EXPRESSION	CONVERTIBILITY	EXPANDABILITY	UNIFORMITY
VALIDITY	DOCUMENTATION	AUGMENTABILITY	SELF-CONTAINEDNESS
PERFORMANCE	UNDERSTANDABILITY	MODIFIABILITY	TIME
RELIABILITY	CLARITY	TESTABILITY	
AVAILABILITY	LEGIBILITY	ACCOUNTABILITY	
ACCURACY	SELF-DESCRIPTIVENESS	COST	
ROBUSTNESS	MAINTAINABILITY	PORTABILITY	
PRECISION	STABILITY	TRANSFERABILITY	
TOLERANCE	MANAGEABILITY	COMPATIBILITY	
EFFICIENCY	CONCISENESS	REUSABILITY	
INTEGRITY	REPAIRABILITY	GENERALITY	
SECURITY	SERVICEABILITY	UTILITY	
PRIVACY			

INSTRUCTOR NOTES

THESE OBJECTIVES ARE COMMON TO ALL SYSTEM DEVELOPMENTS. THERE ARE OTHER OBJECTIVES, SUCH AS BUILDING A SECURE (HANDLING OF CLASSIFIED DATA) SYSTEM THAT ONLY APPLY IN SPECIAL SITUATIONS. BUILDING A SYSTEM WITH HUMAN SAFETY CONSIDERATIONS WOULD BE ANOTHER SPECIAL OBJECTIVE. THESE SPECIAL OBJECTIVES IMPLY ADDITIONAL GOALS.

FOR EACH OF THESE FUNDAMENTAL OBJECTIVES ON THE NEXT THREE VIEWGRAPHS WE ARE GOING TO EXAMINE ITS IMPLICATIONS AND THE SPECIFIC SOFTWARE ENGINEERING GOALS THAT SUPPORT THESE OBJECTIVES.

FUNDAMENTAL SOFTWARE ENGINEERING OBJECTIVES

- BUILD A SYSTEM THAT MEETS USER EXPECTATIONS
- BUILD A SYSTEM THAT CAN ACCOMMODATE CHANGE
- BUILD A SYSTEM WITHIN AVAILABLE RESOURCES

INSTRUCTOR NOTES

THIS IS CERTAINLY AN OBJECTIVE THAT ALL SYSTEMS MUST MEET, BUT SURPRISINGLY FEW DO. MOST OF OUR FAILURE IS IN FRONT-END ERRORS OF NOT PROPERLY UNDERSTANDING WHAT THE SYSTEM WILL BE USED FOR AND TRANSLATING THESE NEEDS INTO A SPECIFICATION THAT CAN BE REVIEWED AND APPROVED BY THE END-USER AND THEN USED BY THE IMPLEMENTORS TO GUIDE THE DEVELOPMENT. IN A TYPICAL SYSTEM IMPLEMENTATION THERE ARE SEVERAL LEVELS OF DOCUMENTATION (I.E., SYSTEM SPEC, SOFTWARE SPEC, DESIGN SPEC, ETC.). EACH LEVEL OF DOCUMENTATION IS A NEW OPPORTUNITY FOR THE SYSTEM TO DIVERGE FROM END-USER NEEDS. PRODUCING A <u>CORRECT</u> SYSTEM REQUIRES <u>UNDERSTANDABLE</u> DOCUMENTS THAT ALLOW THE END-USER'S DESCRIPTION OF THE SYSTEM TO BE <u>TRACED</u> THROUGH EACH LEVEL DOWN TO THE IMPLEMENTATION.

RELIABILITY CAN BE ASSURED IF THE REQUIREMENTS AT EACH LEVEL CAN BE <u>TESTED</u> OR <u>VERIFIED</u>. THIS IS MORE THAN JUST BEING ABLE TO BE UNDERSTOOD. <u>TESTABILITY</u> REQUIRES THAT WE BE PRECISE AND SPECIFIC AND DEAL WITH MEASURABLE QUANTITIES. VERIFIABILITY REQUIRES ORGANIZATIONAL AIDS AND APPROACHES THAT ALLOW US TO IDENTIFY INCONSISTENCIES, INCOMPLETENESS AND OTHER ERRORS.

OBJECTIVE #1 - BUILD A SYSTEM THAT MEETS USER EXPECTATIONS

- SYSTEM MUST BE CORRECT AND RELIABLE
- CORRECTNESS MUST BE TRACEABLE FROM SPECIFICATION, TO DESIGN, AND TO CODE
- INSURING CORRECTNESS REQUIRES UNDERSTANDABILITY OF SYSTEM DESCRIPTIONS
- RELIABILITY REQUIRES TESTABILITY AND VERIFIABILITY

INSTRUCTOR NOTES

VERY FEW SOFTWARE SYSTEMS ARE DEVELOPED AND DEPLOYED AND THEN LEFT UNCHANGED FOR THE REMAINDER OF THEIR USEFUL LIFE. MOST OF THE DOLLARS SPENT ON ANY WEAPON SYSTEM ARE SPENT IN MODIFYING IT TO MEET NEW OR CHANGING REQUIREMENTS. EVEN FOR SYSTEMS THAT REMAIN UNCHANGED, THERE IS USUALLY THE NEED TO MAKE SOME MODIFICATIONS TO CORRECT ERRORS FOUND DURING THE INITIAL DEVELOPMENT. TYPICALLY IT COSTS FAR MORE PER LINE OF CODE TO MODIFY SOFTWARE THAN IT DOES FOR THE INITIAL DEVELOPMENT - THIS IS BECAUSE THE SOFTWARE WAS NOT DESIGNED TO BE EASY TO CHANGE.

EACH OF THE FOLLOWING FOUR GOALS, WHILE RELATED, HAS A SLIGHTLY DIFFERENT EMPHASIS:

- MAINTAINABILITY - EMPHASIZES EASE OF FINDING AND FIXING ERRORS - USUALLY SMALL IN SCALE
- MODIFIABILITY - EMPHASIZES THE EASE OF ACCOMMODATING CHANGES OF ANY SIZE OR TYPE
- TRANSPORTABILITY - EMPHASIZES THE EASE OF MOVING THE CODE TO ANOTHER COMPUTER SYSTEM (MACHINE INDEPENDENCE)
- REUSABILITY - EMPHASIZES THE ABILITY TO USE COMPONENTS ON A NEW (AND POSSIBLY DIFFERENT) APPLICATION

WE MUST ALWAYS BE ABLE TO UNDERSTAND SOMETHING BEFORE WE CAN CHANGE IT OR REUSE IT.

OBJECTIVE #2 - BUILD A SYSTEM THAT CAN ACCOMMODATE CHANGE

- OVER 75% OF SOFTWARE DOLLARS SPENT ON MAINTENANCE
- CHANGE OFTEN NECESSARY TO ACHIEVE INITIAL OPERATIONAL CAPABILITY
- CHANGE OCCURS OVER THE LIFE-CYCLE AS ENVIRONMENT CHANGES
- THIS OBJECTIVE RELATED TO SEVERAL GOALS:
  - <u>MAINTAINABILITY</u>
  - <u>MODIFIABILITY</u>
  - <u>TRANSPORTABILITY</u>
  - <u>REUSABILITY</u>
- APPLIES TO DOCUMENTS AND CODE
- IMPLIES AN UNDERLYING GOAL OF <u>UNDERSTANDABILITY</u>

INSTRUCTOR NOTES

EFFICIENCY IS ONE GOAL THAT CAN OFTEN BE IMPROVED LATE IN THE LIFE-CYCLE, PARTICULARLY IF THE SOFTWARE HAS MET ITS MODIFIABILITY GOAL. WE CAN TUNE THE 20% OF SOFTWARE THAT TAKES 80% OF THE RESOURCES DURING THE SYSTEM TEST PHASE.

THE MOST OBVIOUS EXAMPLES OF PRODUCTIVITY ENHANCING TOOLS ARE HIGH-LEVEL LANGUAGES, BUT GOOD PLANNING CAN PROBABLY SAVE MORE MONEY AND SCHEDULE TIME THAN USING A HIGH LEVEL LANGUAGE.

OBJECTIVE #3 - BUILD A SYSTEM WITHIN AVAILABLE RESOURCES

- MUST ADDRESS BOTH RUNTIME AND IMPLEMENTATION RESOURCES
- RUNTIME RESOURCES REQUIRE EFFICIENCY
  - BOTH STORAGE AND CPU
- IMPLEMENTATION CONSTRAINTS REQUIRE PRODUCTIVITY
- EFFICIENCY CAN OFTEN BE IMPROVED LATE IN DEVELOPMENT
  - 20% OF CODE USES 80% OF CPU TIME
- PRODUCTIVITY IS ENHANCED BY METHODS, TOOLS, PLANNING

INSTRUCTOR NOTES

AT THE START OF A SYSTEM DEVELOPMENT IT IS IMPORTANT TO ESTABLISH THE RELATIVE IMPORTANCE OF THE SYSTEM'S OBJECTIVES - IS THE ABILITY TO RESPOND TO CHANGE MORE IMPORTANT THAN GETTING IT RIGHT IN THE FIRST PLACE? THE RELATIVE IMPORTANCE OF OBJECTIVES ALLOWS US TO TRADE-OFF THE INDIVIDUAL GOALS. FORTUNATELY MANY GOALS ARE MUTUALLY COMPATIBLE. THE TOUGHEST TRADE-OFF IS OFTEN WITH EFFICIENCY. AS A GENERAL RULE OF THUMB WE SHOULD NOT BE OVERLY CONCERNED WITH EFFICIENCY EARLY ON. IT IS HARD EARLY ON TO PREDICT SYSTEM BOTTLENECKS AND MAJOR RESOURCE USERS. BY STRESSING EFFICIENCY TOO EARLY WE LOSE UNDERSTANDABILITY AND MODIFIABILITY AND OFTEN DEGRADE EFFICIENCY BECAUSE WE LOSE OUR ABILITY TO TUNE THE SYSTEM.

SOFTWARE ENGINEERING GOAL CONFLICTS

- SOME GOALS ARE MUTUALLY COMPATIBLE
  - UNDERSTANDABILITY, MODIFIABILITY

- CONFLICTING GOALS REQUIRE TRADE-OFFS
  - EFFICIENCY VS. UNDERSTANDABILITY
  - PRODUCTIVITY VS. EFFICIENCY
  - RELIABILITY VS. PRODUCTIVITY

- TRADE-OFFS BETWEEN GOALS BASED ON SYSTEM OBJECTIVES

- TRADE-OFFS MUST BE AGREED TO BY ALL PARTICIPANTS
  - USER, MANAGER, IMPLEMENTORS

INSTRUCTOR NOTES

SECTION 4

SOFTWARE ENGINEERING PRINCIPLES

INSTRUCTOR NOTES

THIS SECTION DEFINES SOME BASIC ENGINEERING PRINCIPLES THAT WORK TOWARD ACHIEVING OUR FUNDAMENTAL GOALS.

NOT ALL OF THESE PRINCIPLES ADDRESS THE ENTIRE LIFE-CYCLE; SOME ARE LIMITED TO FRONT-END CONSIDERATIONS, AND OTHERS APPLY ONLY TO DESIGN AND IMPLEMENTATION.

SOME OF THESE PRINCIPLES ARE WIDELY ACCEPTED AND USED; OTHERS ARE STILL CONTROVERSIAL AND, WHILE INTUITIVELY APPEALING, HAVE NOT BEEN WIDELY USED AND PROVEN "UNDER FIRE."

AS WE REVIEW THE SPECIFIC TOOLS AND METHODS DURING LATER SECTIONS OF THIS COURSE, WE WILL IDENTIFY AND DISCUSS THE SPECIFIC PRINCIPLES UPON WHICH EACH OF THESE TOOLS OR METHODS IS BASED.

SOFTWARE ENGINEERING PRINCIPLES

- ARE THE TECHNIQUES FOR ATTAINING SOFTWARE GOALS
- APPLY TO DIFFERENT PHASES OF LIFE-CYCLE
- REPRESENT AN EVOLVING CONSENSUS
  - NOT ALL PROVEN OR ACCEPTED THROUGHOUT INDUSTRY
- WILL RELATE TO SPECIFIC TOOLS AND METHODS DURING LATER SECTIONS OF THE COURSE

INSTRUCTOR NOTES

GO THROUGH EACH PRINCIPLE AND ITS BRIEF DEFINITION. EACH IN TURN WILL BE COVERED IN MORE DETAIL.

SUPPRESSION OF DETAIL MEANS THAT YOU IDENTIFY AND OMIT -- FOR THE MOMENT -- NONESSENTIAL DETAILS.

UNDERLYING PRINCIPLES

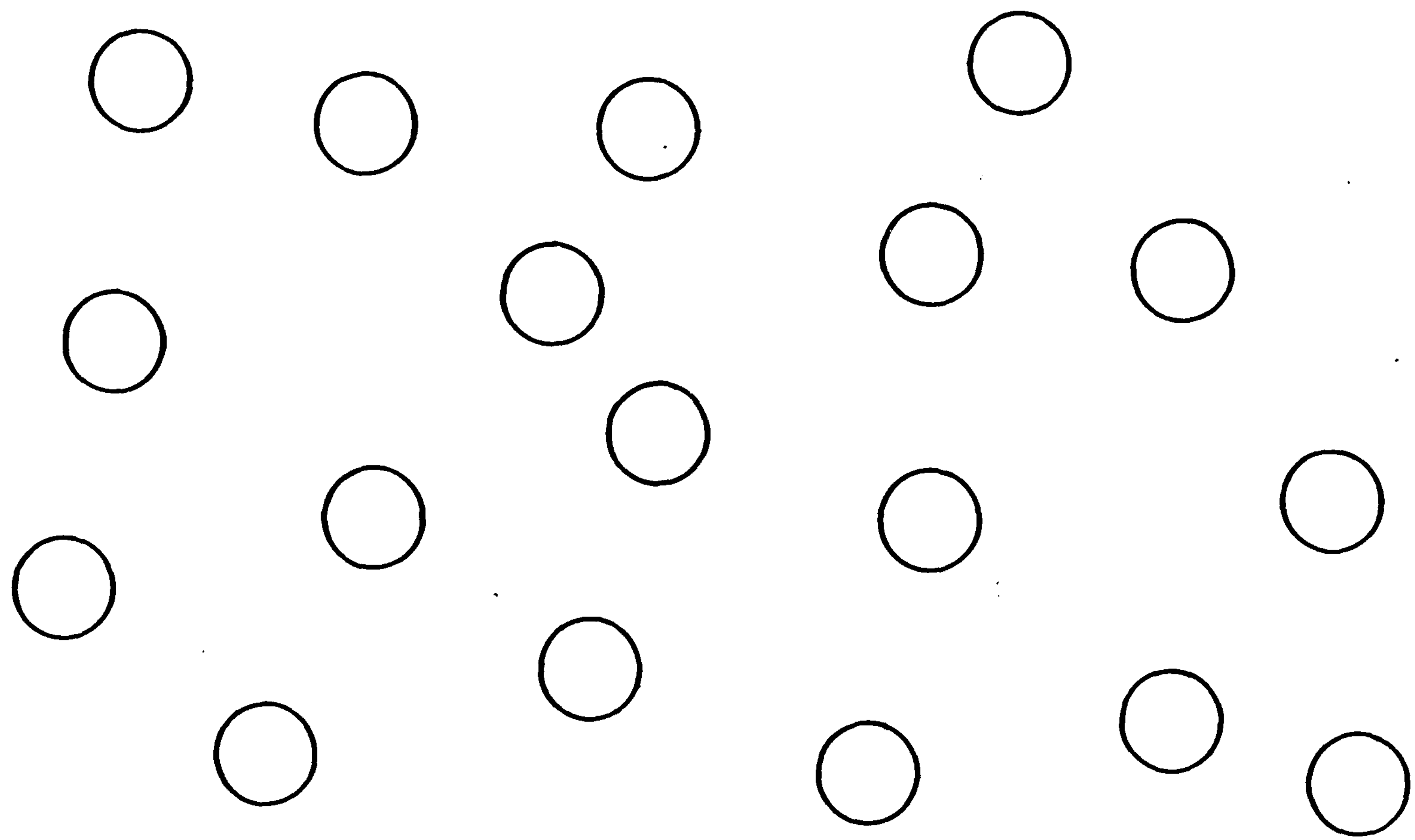
- STRUCTURING -- RELATING PARTS AND SUBPARTS
- MODULARITY -- RATIONAL STRUCTURING
- ABSTRACTION -- SUPPRESSION OF DETAIL
- HIDING -- MAKING SOME OF THE "HOWS" UNKNOWN TO OTHER PARTS OF THE SYSTEM
- SEPARATION OF CONCERNS -- PHYSICALLY COLLECTING RELATED THINGS AND SEPARATING UNRELATED THINGS
- UNIFORMITY -- CONSISTENCY
- FORMALISM -- AVOID PROSE, BE PRECISE AND CONCISE

INSTRUCTOR NOTES

<u>INSTRUCTOR</u>:

HOLD EACH SLIDE UP FOR EXACTLY 2 SECONDS. HAVE CLASS TRY TO COUNT CIRCLES, WRITING DOWN THEIR ANSWERS. EVERYONE SHOULD COME CLOSER TO THE CORRECT ANSWER (18) ON THE SECOND TRY, BECAUSE SHADING ENCODED INDIVIDUAL CIRCLES INTO COUNTABLE GROUPS.

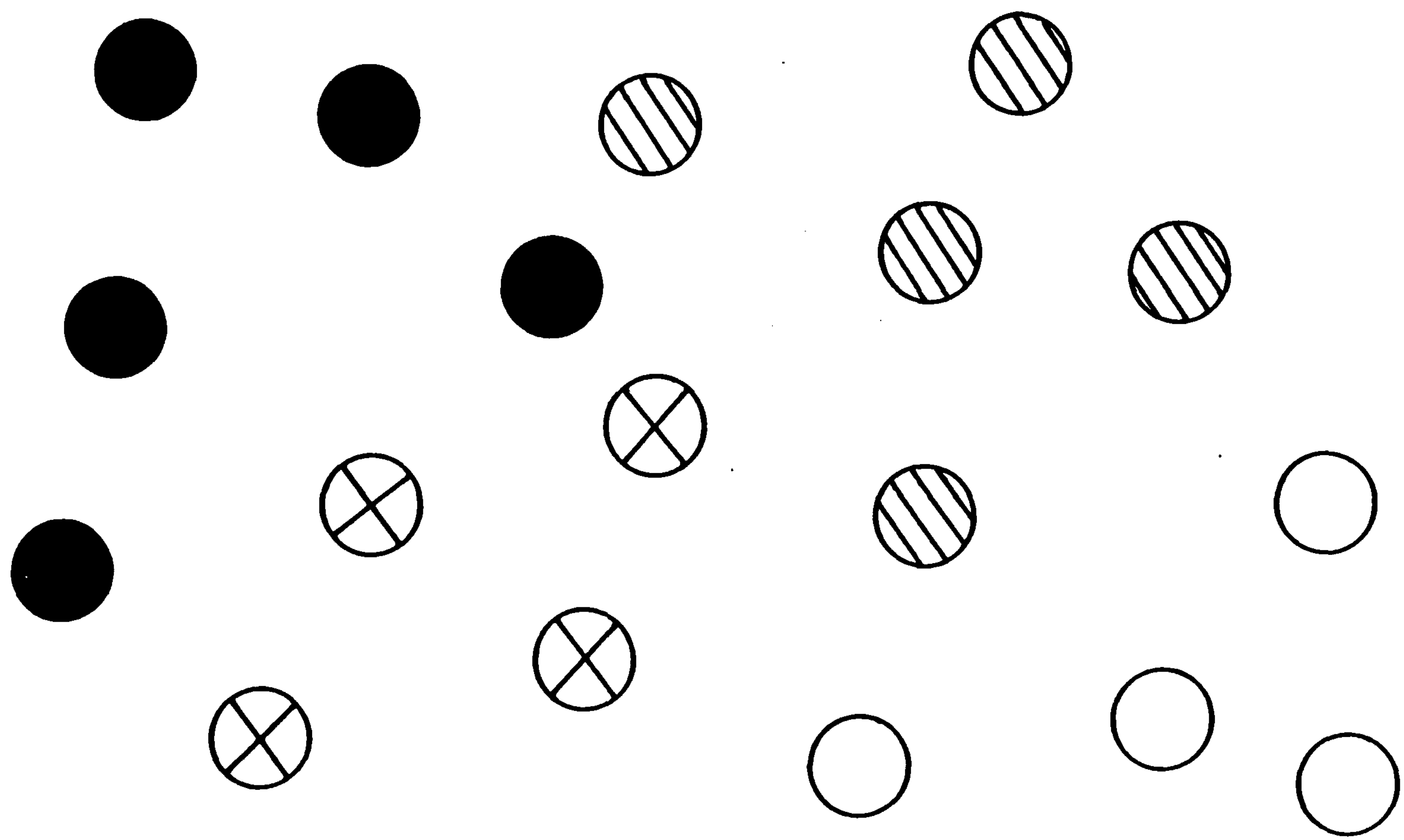
STRUCTURING



INSTRUCTOR NOTES

REFER TO PREVIOUS INSTRUCTOR NOTE.

NOW, COUNT THE CIRCLES



INSTRUCTOR NOTES

AS THE PREVIOUS EXAMPLE POINTED OUT, INTRODUCING SOME STRUCTURING -- IN THIS CASE DIVIDING THE OBJECTS INTO CLASSES -- HAS ENHANCED OUR UNDERSTANDABILITY - AT LEAST AS FAR AS COUNTING THE OBJECTS GOES.

THERE ARE VARIOUS TYPES OF STRUCTURING WHICH ARE APPLICABLE IN DIFFERENT SITUATIONS. SOME ORGANIZATIONAL STRUCTURING IS USED IN DEVELOPING DOCUMENTS - HIERARCHY AND PARTITIONING INTO CLASSES OR CATEGORIES ARE EXAMPLES. OTHER FORMS OF STRUCTURING ARE USED IN IMPLEMENTATIONS (DESIGN AND CODE); EXAMPLES ARE NETWORKS, TASKING STRUCTURES, AND STRUCTURED LANGUAGES.

ANY STRUCTURING MECHANISM WORKS BY ESTABLISHING RULES AND CONVENTIONS. THESE CONVENTIONS USUALLY RESULT IN SOME LOSS OF FLEXIBILITY AND GENERALITY. WE DO PAY SOME PRICE FOR THE INCREASE IN UNDERSTANDABILITY - EVERY ONCE IN A WHILE THE STRUCTURE GETS IN THE WAY.

STRUCTURING

- STRUCTURING REDUCES COMPLEXITY, AIDS UNDERSTANDING

- STRUCTURING COMES IN DIFFERENT FORMS

  - HIERARCHICAL
  - NETWORK
  - PROCESS/TASKING
  - "STRUCTURED PROGRAMMING LANGUAGES"
  - CLASSES, TYPES, CATEGORIES

- ESTABLISHES RULES AND CONVENTIONS

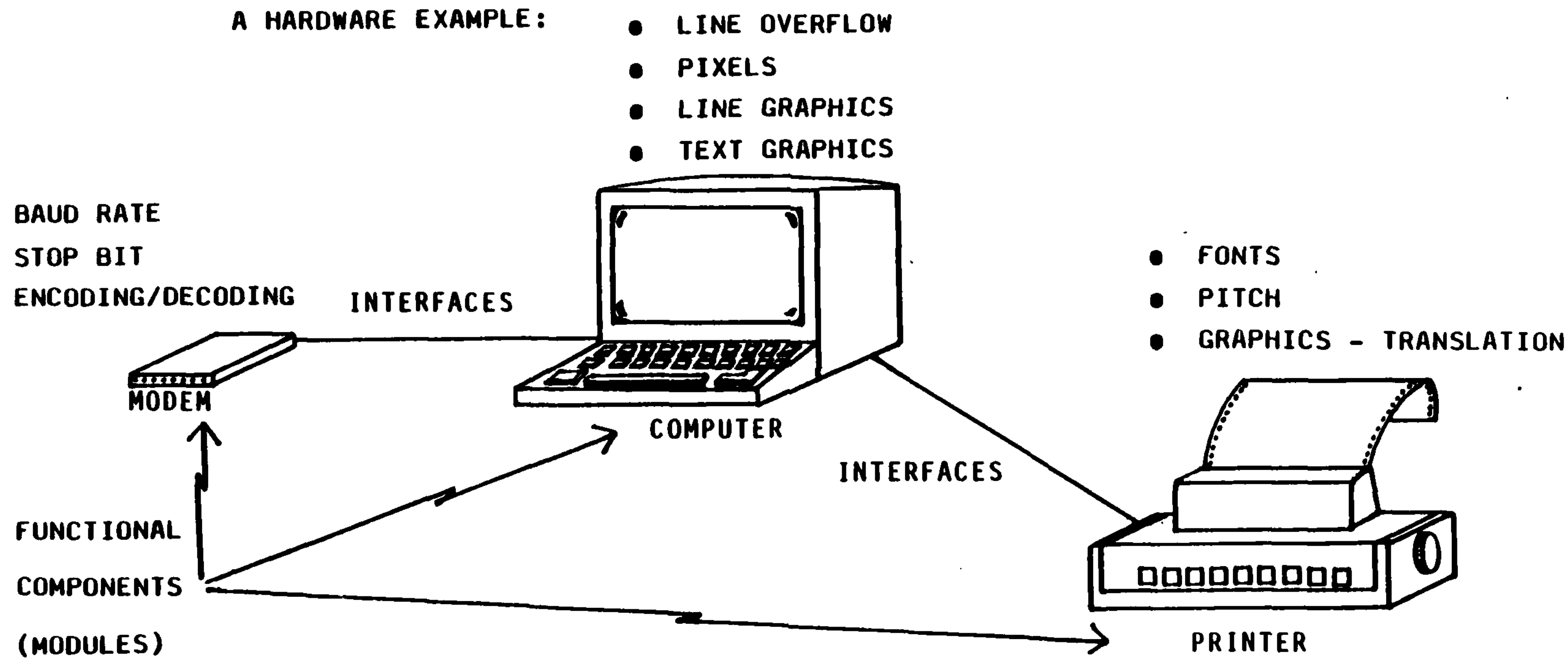
  - REDUCES FLEXIBILITY

INSTRUCTOR NOTES

STRUCTURING BREAKS A SYSTEM INTO PARTS, BUT MODULARITY GIVES PURPOSE TO THE STRUCTURING.

MODULARITY GOES HAND-IN-HAND WITH THE DEVELOPMENT OF EXPLICIT INTERFACES BETWEEN THE MODULES. THESE EXPLICIT INTERFACES ALLOW THE SEPARATE MODULES TO BE DEVELOPED INDEPENDENTLY, POSSIBLY IN PARALLEL.

MODULARITY



- EACH MODULE HAS A SPECIFIC FUNCTION WITH WELL-DEFINED INTERFACES

INSTRUCTOR NOTES

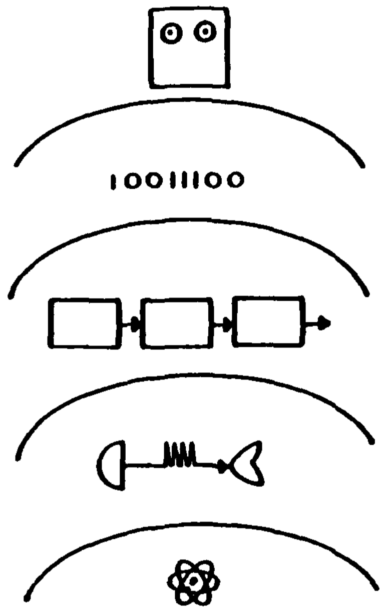
ABSTRACTION AIDS UNDERSTANDING BY SUPPRESSING UNNECESSARY DETAIL WHICH ALLOWS ESSENTIAL CHARACTERISTICS TO STAND OUT. FOR THE HIGH LEVEL LANGUAGE TO WORK, ALL THE OTHER LEVELS MUST WORK - WE TRUST THIS TO BE TRUE, BUT WE DON'T HAVE TO KNOW ANYTHING ABOUT THESE OTHER LEVELS.

THIS SAME ABSTRACTION OCCURS IN MANY OTHER SITUATIONS - WE DON'T NEED TO KNOW HOW A STEERING WHEEL WORKS TO DRIVE A CAR OR A BOAT. WE LEARN TO ABSTRACT THE DETAILS AND ASSOCIATE CLOCKWISE TURNING OF THE WHEEL WITH A RIGHT TURN.

ABSTRACTION

```
Procedure Sort is
    ...
begin
    ...
end Sort;
```

- HIGH LEVEL LANGUAGE LEVEL



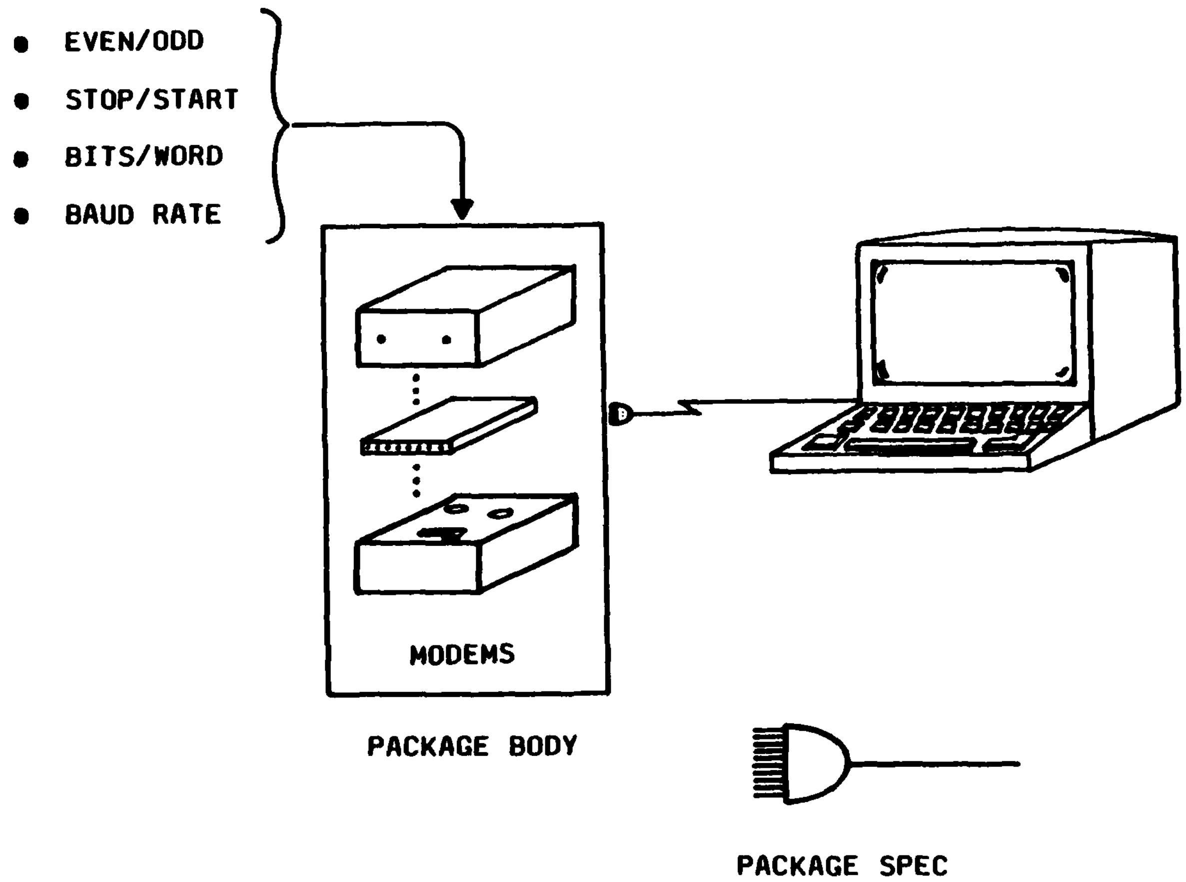
- MACHINE LANGUAGE LEVEL
- REGISTER TRANSFER LEVEL
- CIRCUIT LEVEL
- PHYSICS LEVEL

- EXPRESSES A PARTICULAR USAGE VIEWPOINT
- SUPPRESSES UNRELATED DETAIL

STRESS THAT HIDING IS A PRINCIPLE THAT WE ALL USE INTUITIVELY. HARDWARE ENGINEERS HAVE DONE A MUCH BETTER JOB OF USING IT THAN HAVE SOFTWARE ENGINEERS. WHENEVER WE PRODUCE MODULES WITH WELL DEFINED INTERFACES, WE ARE USING HIDING - WE HAVE HIDDEN EVERYTHING THAT IS NOT EXPLICITLY INCLUDED IN THE INTERFACE.

THE MODEM EXAMPLE IS A CLASSIC HARDWARE EXAMPLE OF HIDING. WE KNOW NOTHING ABOUT THE INTERNAL DESIGN OF THE MODEM - VERY DIFFERENT IMPLEMENTATION APPROACHES CAN BE USED AND WE STILL CAN REPLACE ONE MODEM WITH A DIFFERENT ONE BECAUSE THEY BOTH MEET THE SAME INTERFACE - SHOWN HERE GRAPHICALLY, AS MEETING THE SAME PACKAGE SPEC. THE PACKAGE BODY - THE IMPLEMENTATION - CAN BE COMPLETELY DIFFERENT.

HIDING



- THE INTERFACE "HIDES" THE BRAND OR TYPE OF MODEM.

INSTRUCTOR NOTES

THESE PRINCIPLES ARE NOT USUALLY USED INDEPENDENTLY BUT IN FACT ARE RELATED AND USED TOGETHER IN MODERN SE METHODOLOGIES. MODULARITY, ABSTRACTION, AND HIDING ARE CLOSELY RELATED AND FORM THE BASIS FOR THE DESIGN APPROACH EMBODIED IN THE SOFTWARE COST PRODUCTION PROJECT. THIS PROJECT WAS LED BY AND IS BASED ON EARLIER RESEARCH BY DAVID PARNAS. WE WILL DISCUSS THIS METHODOLOGY IN THE METHODS AND TOOLS PART OF THE COURSE. (REVIEW THAT MATERIAL FOR A BETTER UNDERSTANDING.) BRIEFLY THE SCRP USES AS THEIR MODULARIZATION CRITERIA EXPECTED CHANGE. THE ANALYSTS AND DESIGNERS ATTEMPT TO PREDICT WHAT ASPECTS OF THE SYSTEM ARE LIKELY TO CHANGE - THIS IS BASED ON PAST EXPERIENCE AND SOME KNOWLEDGE OF FUTURE PLANS. THINGS LIKE CHANGES TO SENSORS AND EXTERNAL DEVICES ARE EXAMPLES OF LIKELY CHANGES. EACH MODULE IN THE SYSTEM IS DESIGNED AROUND EACH INDEPENDENT CHANGE OR GROUP OF RELATED CHANGES. THE MODULE "HIDES" THE IMPACT OF THE CHANGE BY PRESENTING AN "ABSTRACT" INTERFACE TO THE REST OF THE SYSTEM WHICH CAN STILL BE MET EVEN IF THE EXPECTED CHANGE TAKES PLACE. IN OTHER WORDS, THE INTERFACE ONLY PRESENTS THE ESSENTIAL CHARACTERISTICS OF A SENSOR THAT IS LIKELY TO BE REPLACED. IT HIDES ANY ARBITRARY DETAILS THAT MAY VARY BETWEEN DIFFERENT MAKES OR VERSIONS.

MODULARITY / ABSTRACTION / HIDING

- USED TOGETHER TO DEFINE ABSTRACT MODULE SPECIFICATIONS
- KEY COMPONENT OF THE SOFTWARE COST REDUCTION PROJECT (SCRP) (PARNAS) METHODOLOGY
- MODULARIZATION STRATEGY BASED ON EXPECTED CHANGE
- A MODULE HIDES EFFECTS OF INDEPENDENT CHANGE
- (ABSTRACT) MODULE INTERFACE ONLY MAKES VISIBLE THOSE ASPECTS OF A MODULE UNLIKELY TO CHANGE

INSTRUCTOR NOTES

"SEPARATION OF CONCERNS" IS ALSO AN IMPORTANT ASPECT OF THE SCRP METHODOLOGY. THE TERM WAS COINED BY DYKSTRA (I BELIEVE) IN HIS BOOK, <u>A DISCIPLINE OF PROGRAMMING</u>. THE GOAL IS TO INSURE THAT THE IMPACT OF CHANGE IS MINIMIZED AND THAT WE ALWAYS KNOW WHERE TO GO TO FIND THE ANSWER TO QUESTIONS. FOR EXAMPLE, HARDWARE INTERFACES ARE DESCRIBED WITHOUT MAKING ASSUMPTIONS ABOUT THE FUNCTIONALITY OF THE SYSTEM; THE HARDWARE INTERFACE DESCRIPTION WOULD REMAIN UNCHANGED EVEN IF THE FUNCTION CHANGED. THIS SEPARATION OCCURS AT MULTIPLE LEVELS; AT A HIGH LEVEL WE SEPARATE FUNCTIONS FROM BEHAVIOR FROM HARDWARE INTERFACE (I/O MAPPING) FROM TIMING, ETC. WITHIN EACH SUBJECT WE USE A STANDARD TEMPLATE TO FURTHER SEPARATE THE DETAILS IN EACH SUBJECT AREA. ALLOWS FOR EASE OF ACCESSING INFORMATION, IDENTIFIES AREAS NEEDING CHANGE, AND CREATES A "FILL-IN-THE-BLANKS" APPROACH TO WRITING SPECS ONCE THE TEMPLATES HAVE BEEN DEFINED. THIS LEADS TO AN UNSTATED PRINCIPLE "STATE QUESTIONS BEFORE TRYING TO ANSWER THEM."

SEPARATION OF CONCERNS

- ALSO KNOWN AS LOCALIZATION

- GOALS
  - REDUCE THE IMPACT OF CHANGE
  - ALWAYS KNOW WHERE TO FIND ANSWERS TO QUESTIONS

- SEPARATE (FOR EXAMPLE)
  - FUNCTIONALITY FROM BEHAVIOR
  - INPUT/OUTPUT BIT REPRESENTATIONS FROM LOGICAL MEANING
  - TIMING
  - ACCURACY
  - EXPECTED CHANGES
  - INPUT/OUTPUT MAPPING FROM FUNCTIONALITY

- USE STANDARD TEMPLATES FOR ADDITIONAL LEVEL OF SEPARATION

INSTRUCTOR NOTES

THIS IS AN EXAMPLE OF A BLANK SCRP INPUT DATA FORM. POINT OUT THAT AS AN EXAMPLE OF HIGH LEVEL SEPARATIONS OF CONCERNS THERE IS NO MENTION OF HOW THIS INPUT IS USED - WHAT FUNCTIONS DEPEND ON IT, OR HOW IT EFFECTS BEHAVIOR.

AT A LOWER LEVEL IT PROVIDES A SET OF QUESTIONS TO ASK AND ANSWER ABOUT EACH HARDWARE INTERFACE. AT THIS LOW LEVEL THE "ESSENTIAL CHARACTERISTICS" SUCH AS UNITS AND RANGE OR THE POSSIBLE VALUES (ON OR OFF) FOR VALUE ENCODING ARE SEPARATED FROM THE DATA REPRESENTATION (I.E., THE MAPPING TO BITS) AND THE INSTRUCTION SEQUENCE NEEDED TO ACQUIRE THE DATA.

I/O DATA TEMPLATE EXAMPLE

INPUT DATA ITEM:

ACRONYM:

HARDWARE:

DESCRIPTION:

CHARACTERISTICS OF VALUES:

UNITS: RANGE: ACCURACY: RESOLUTION:	OR	VALUE ENCODING:

INSTRUCTION SEQUENCE:

DATA REPRESENTATION:

TIMING CHARACTERISTICS:

INSTRUCTOR NOTES

THIS PRINCIPLE IS MORE GENERAL AND HIGH LEVEL THAN THE OTHERS. SPECIFIC EXAMPLES ARE SUCH THINGS AS NAMING CONVENTIONS WHICH SUPPORT UNDERSTANDABILITY AND CONFIGURATION MANAGEMENT. SPECIFIC Ada EXAMPLES OF CONSTRUCTS WHICH SUPPORT UNIFORMITY ARE DATA TYPES, OVERLOADING OF OPERATORS AND GENERICS (WHICH ALLOWS ALL SIMILAR OPERATIONS TO HAVE THEIR SIMILARITIES MADE EXPLICIT).

DEFINING AND USING A LIMITED TASKING MODEL, AS Ada HAS DONE, IS ANOTHER EXAMPLE OF UNIFORMITY.

THE USE OF STANDARD TEMPLATES AS DISCUSSED UNDER SEPARATION OF CONCERNS IS ANOTHER EXAMPLE OF UNIFORMITY.

UNIFORMITY

- USE OF DATA TYPES, OVERLOADING, GENERICS
- CONSISTENCY OF TERMINOLOGY (NAMES)
- USE OF LIMITED DESIGN STRUCTURES - TASKING MODEL
- USE OF STANDARD TEMPLATES AND TABLES

INSTRUCTOR NOTES

THE PRINCIPLE OF FORMALISM COMES IN SEVERAL FLAVORS. TO SOME IT MEANS AN APPROACH BASED ON MATHEMATICAL TECHNIQUES THAT WILL LEAD TO AUTOMATED GENERATION OF CODE AND/OR AUTOMATIC VERIFICATION OF CORRECTNESS. TO OTHERS IT IS REPRESENTED BY CONSISTENT USE OF LABELS AND TEMPLATES WITH SOME RULES ON ALLOWED OPERATORS AND POSSIBLE NEW SYMBOLOGY (I.E., USE OF SPECIAL CHARACTERS). SOME TECHNIQUES USE A FORMAL LANGUAGE (MUCH LIKE A PROGRAMMING LANGUAGE) TO EXPRESS THINGS LIKE REQUIREMENTS - PSC/PSM IS A GOOD EXAMPLE OF THIS TYPE OF FORMALISM.

THE GOAL OF FORMALISM IS TO ALLOW DETAILED, PRECISE, AND CONCISE SPECIFICATION TO BE DEVELOPED THAT CAN BE REVIEWED FOR COMPLETENESS AND CORRECTNESS.

IN GENERAL THESE FORMALISMS ARE A REPLACEMENT FOR PROSE WHICH IS RECOGNIZED TO BE AMBIGUOUS AND OFTEN REDUNDANT.

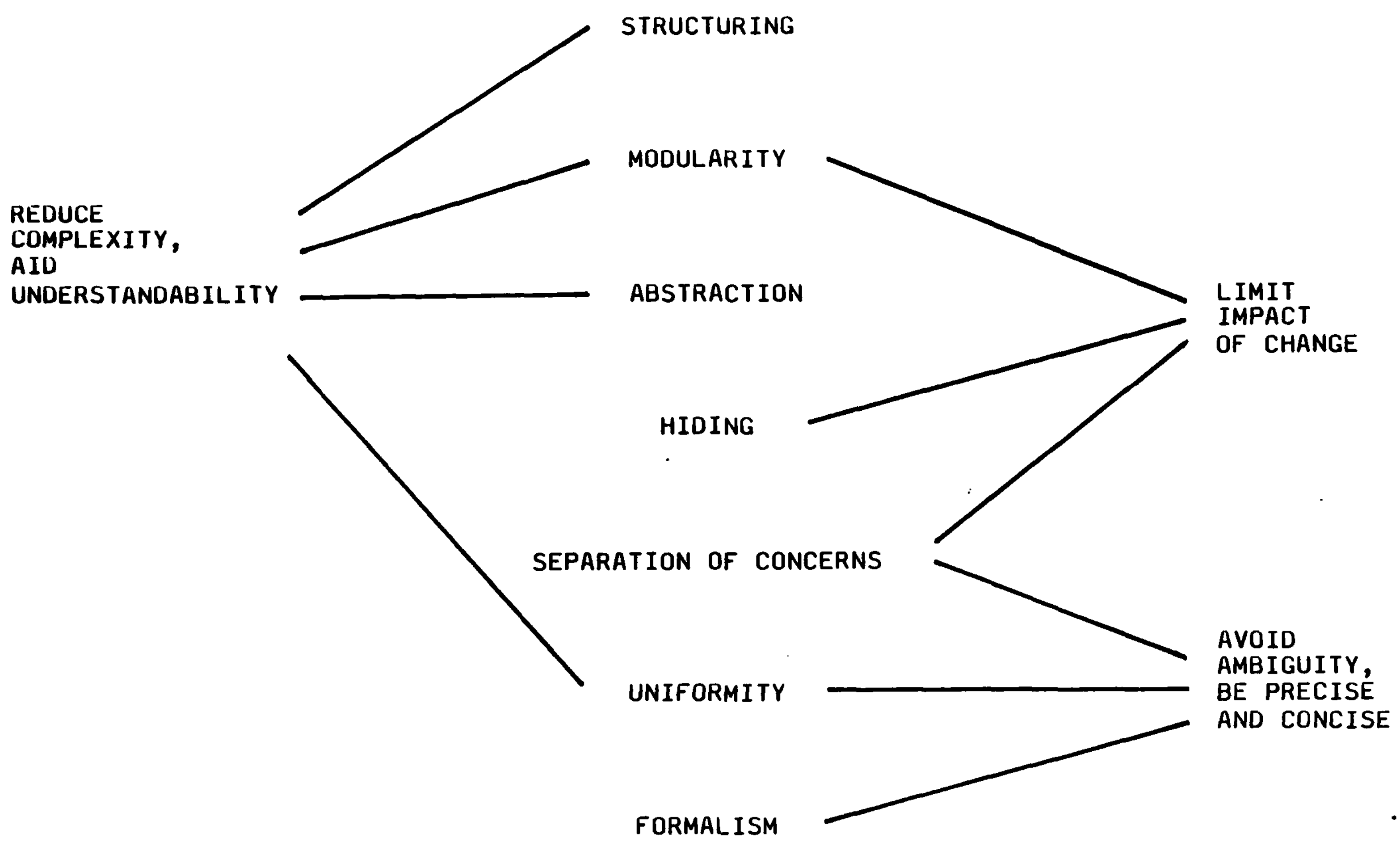
FORMALISM

- SUPPORTS PRECISION, CONCISENESS, COMPLETENESS
- MAY BE BASED ON
  - MATHEMATICS
  - FORMAL LANGUAGE
- MAY USE TABLES, TEMPLATES
- IMPLIES A REDUCTION IN STANDARD PROSE
- LEADS TO
  - MACHINE PROCESSING OF SPECIFICATION
  - MANUAL OR AUTOMATED REVIEW AND VERIFICATION
  - PROOFS OF CORRECTNESS

INSTRUCTOR NOTES

THIS IS JUST A SUMMARY OF THESE PRINCIPLES WITH THE MOST IMPORTANT FOCUS OF THESE PRINCIPLES (IN TERMS OF OUR GOALS AND OBJECTIVES) POINTED OUT.

SOFTWARE ENGINEERING PRINCIPLES



INSTRUCTOR NOTES

- IN THIS PART WE ARE GOING TO COVER

1) SOFTWARE LIFE-CYCLE

2) METHODS AND TOOLS FOR EACH PHASE OF LIFE-CYCLE

3) TESTING

4) SOFTWARE MANAGEMENT

PART III

ACHIEVING SOFTWARE ENGINEERING GOALS

INSTRUCTOR NOTES

THEME: THE SOFTWARE DEVELOPMENT PROCESS CAN BE EXPRESSED IN TERMS OF A SEQUENCE OF PHASES.

PURPOSE: TO PROVIDE THE FRAMEWORK IN WHICH THE METHODOLOGIES WILL BE PRESENTED IN. .

REFERENCE: "A SOFTWARE ENGINEERING ENVIRONMENT FOR THE NAVY", REPORT OF THE NAVMAT SOFTWARE ENGINEERING ENVIRONMENT WORKING GROUP, MARCH 1983.

SECTION 5

THE SOFTWARE LIFE CYCLE

INSTRUCTOR NOTES

THE CONCEPT OF A SOFTWARE DEVELOPMENT LIFE-CYCLE HAS EVOLVED OVER THE LAST 20 YEARS. IT IS VIEWED DIFFERENTLY BY DIFFERENT INDIVIDUALS, ORGANIZATIONS, AND INDUSTRIES, OFTEN BECAUSE THE TYPES OF PRODUCTS THAT ARE PRODUCED ARE RADICALLY DIFFERENT OR BECAUSE OF THE IMPORTANCE OR NON-IMPORTANCE OF THE SOFTWARE USED IN THE PRODUCT DEVELOPMENT.

SOFTWARE LIFE CYCLE

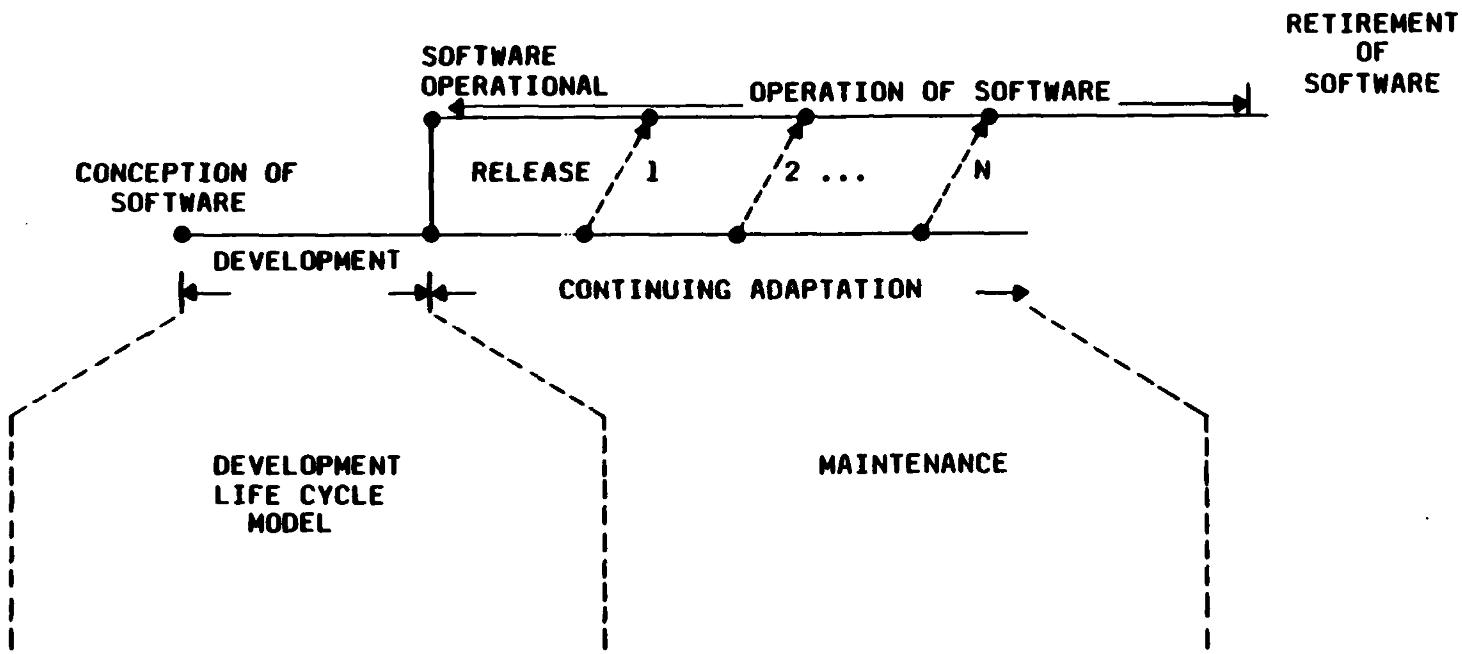
- LIFE CYCLE ORGANIZES THE ACTIVITIES OF BUILDING SOFTWARE
- SOFTWARE DEVELOPMENT IS BROKEN INTO PHASES
- LIFE CYCLE SUMMARIZES SOFTWARE DEVELOPMENT
- THE "LIFE CYCLE" IS NOT STANDARD IN THE INDUSTRY

INSTRUCTOR NOTES

THE TERM "LIFE-CYCLE" PROVIDES US WITH A WAY OF LOOKING AT THE TASKS OF BUILDING A SYSTEM. IT IS NEEDED SO THAT WE CAN TALK ABOUT ASPECTS OF SOFTWARE DEVELOPMENT IN A COMMON LANGUAGE, AND SO THAT WE CAN DETERMINE THE EFFECTS OF CHANGES TO OUR DEVELOPMENT PROCESS.

NOTE THAT SOFTWARE HAS A LIFE OF ITS OWN THAT EXTENDS WELL BEYOND THE CODING OF AN APPLICATION.

THE LIFE OF SOFTWARE



SOURCE: SEEWG REPORT, 1983

INSTRUCTOR NOTES

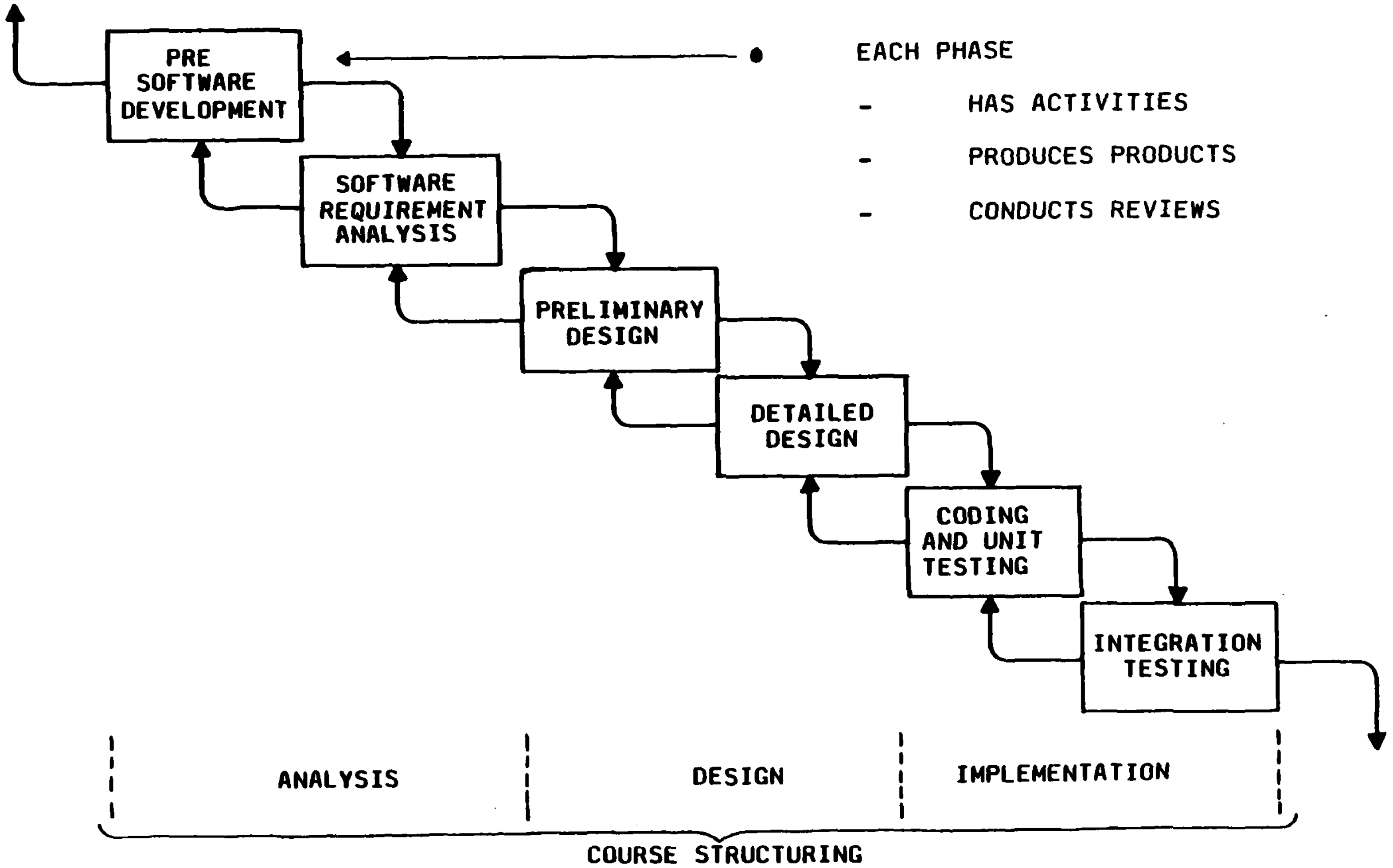
THIS IS REALLY DIVIDING UP THE RESPONSIBILITY OF THE SYSTEM DEVELOPMENT AMONG DIFFERENT GROUPS TO MAKE THE DEVELOPMENT PROCESS TRACTABLE. SOME DIVIDE IT UP INTO 6 PIECES, SOME INTO 4 OR 8 PIECES.

NOTE THE MAPPING TO THE COURSE STRUCTURE. THIS PICTURE IS A MORE CLASSICAL VIEW OF LIFE-CYCLE. THIS MAY (AND PROBABLY WILL) DIFFER FROM THE LIFE-CYCLE MODEL UNDERSTOOD BY EACH STUDENT. THIS IS OFTEN CALLED THE "WATERFALL" MODEL. OUT OF EACH PHASE A DELIVERABLE IS USUALLY PRODUCED, WHICH IS USED AS INPUT INTO THE NEXT PHASE. A REAL SYSTEM OR SOFTWARE DEVELOPMENT IS NEVER THIS CLEAN. THERE EXIST FEEDBACK LOOPS BETWEEN PHASES, THE DELIVERABLES NEED REVIEWS AND CHANGES BEFORE ACCEPTANCE, THE DESIGN PHASE MAY BE SUB-DIVIDED INTO MANY DESIGN PHASES OF INCREASED DETAIL, ETC.

THE KEY WORDS ARE:

- ANALYSIS - THE "WHAT" PHASE
- DESIGN - THE "HOW" PHASE
- IMPLEMENTATION - THE "BUILD" PHASE

DEVELOPMENT LIFE CYCLE MODEL



- EACH PHASE
  - HAS ACTIVITIES
  - PRODUCES PRODUCTS
  - CONDUCTS REVIEWS

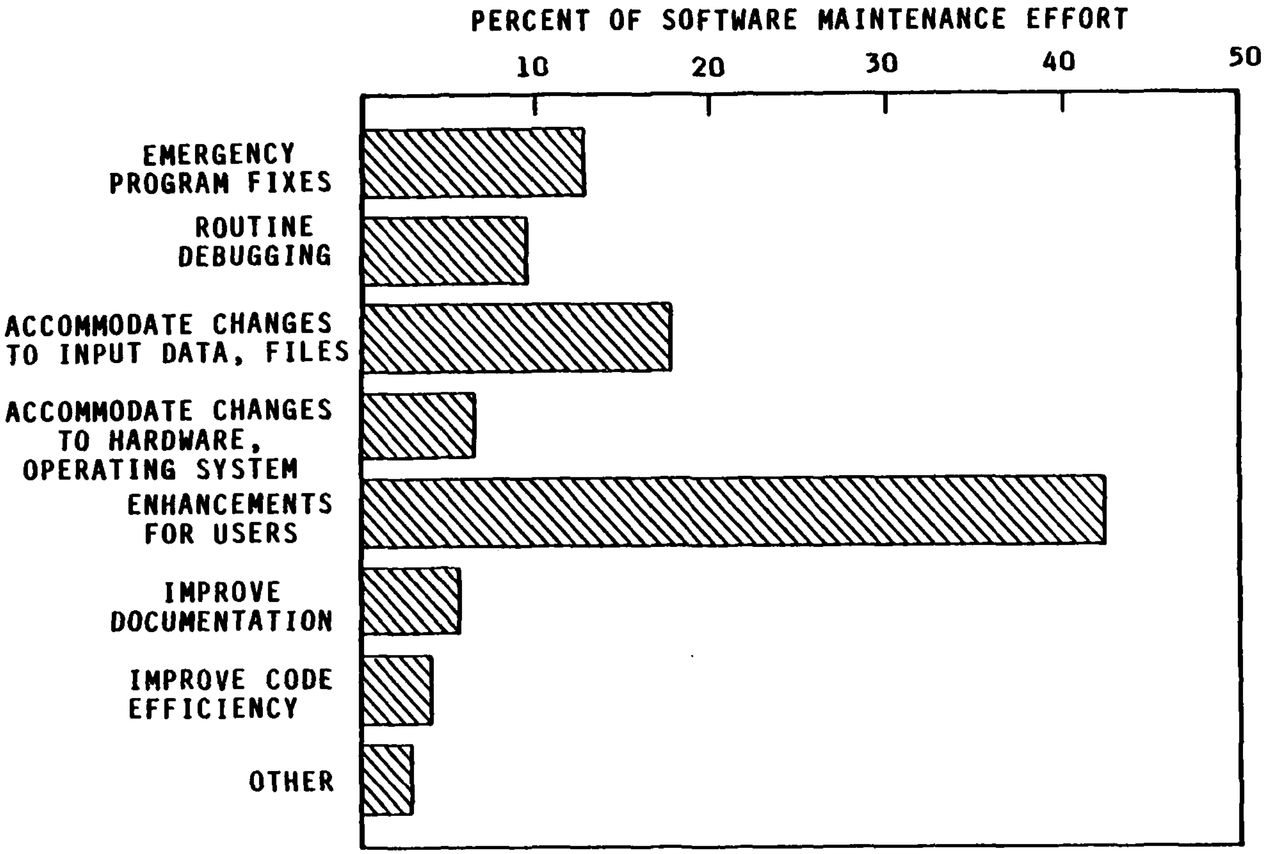
INSTRUCTOR NOTES

MAINTENANCE ISN'T THE BEST TERM FOR WHAT GOES ON AFTER DEVELOPMENT, SOME PEOPLE USE THE WORD

"EVOLUTION"

DUE TO THE EMPHASIS ON ENHANCEMENTS ETC.

SOFTWARE MAINTENANCE ACTIVITIES



SOURCE: BEOHM, SOFTWARE ENGINEERING ECONOMICS, 1983

INSTRUCTOR NOTES

POINT OUT THAT <u>NO</u> MODEL CAN TOTALLY REPRESENT THE REAL WORLD.

THE CRITICAL LIMITATION IS BULLET 3 - CORRECTNESS ANALYSIS.

SHORTCOMINGS OF THIS MODEL

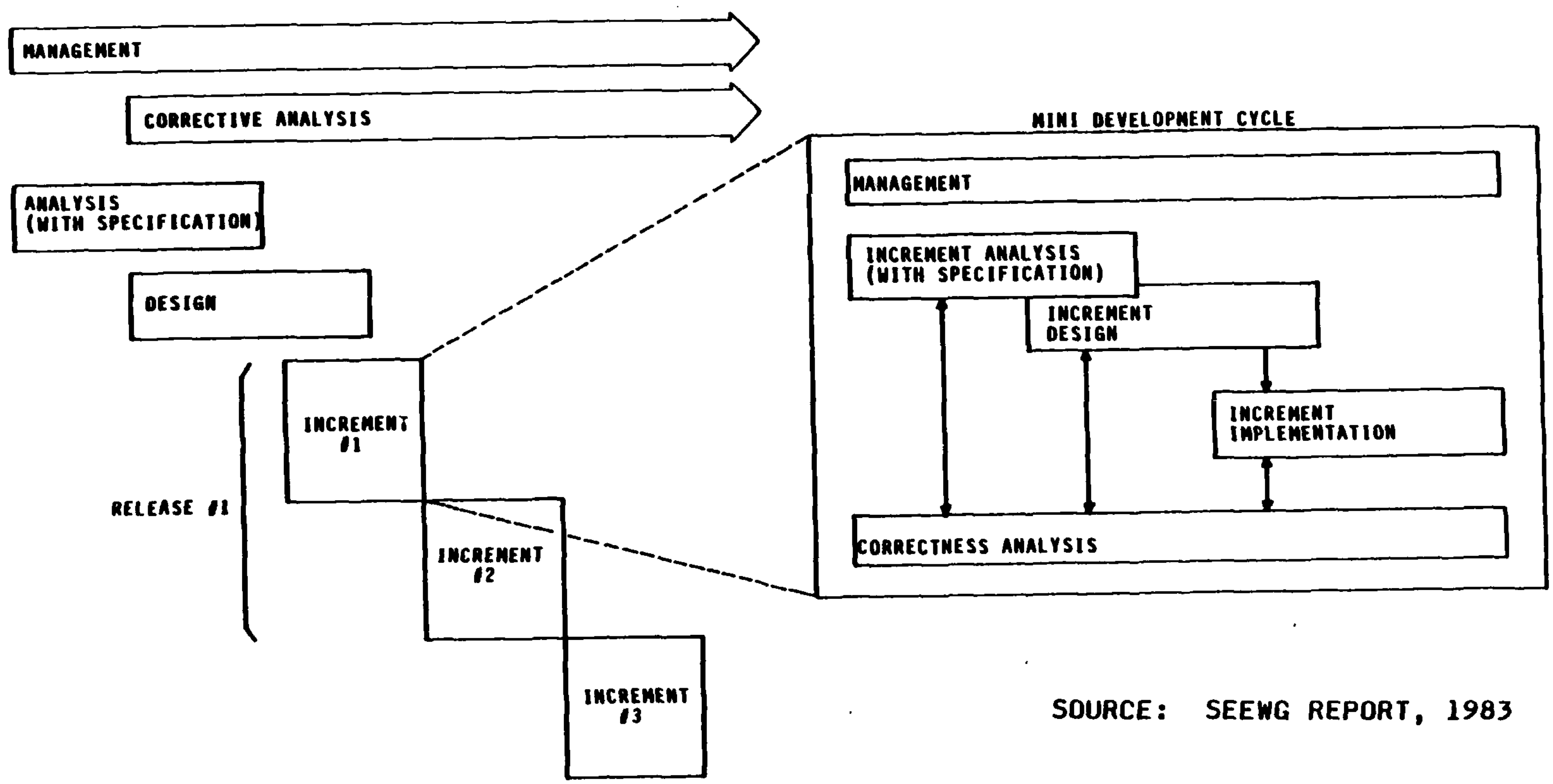
- FINITE ACTIVITY PHASES ARE NOT REALISTIC
  - MOST ACTIVITIES DO NOT HAVE A CLEARLY DEFINED START OR END POINT
- STATIC NATURE OF THE OUTPUTS OF A PREVIOUS ACTIVITY
  - NO OUTPUT CAN BE CONSIDERED COMPLETE; EACH WILL NEED SOME MODIFICATION IN THE NEXT ACTIVITY
  - FEEDBACK PATH HELPS TO A LIMITED DEGREE
- CORRECTNESS ANALYSIS IS PRIMARILY DONE IN TESTING ACTIVITIES
- MANAGEMENT IS NOT EXPLICITLY SHOWN

INSTRUCTOR NOTES

EMPHASIZE CONTINUOUS MANAGEMENT AND CORRECTNESS ANALYSIS.

A MODEL THAT ADDRESSES THE SHORTCOMINGS

- INCREMENTAL DEVELOPMENT
  - BUILD SOFTWARE SYSTEM IN SMALL MANAGEABLE INCREMENTS
  - EACH INCREMENT ADDS NEW FUNCTIONS TO THE SYSTEM



SOURCE: SEEWG REPORT, 1983

INSTRUCTOR NOTES

PURPOSE: TO MOTIVATE FOLLOWING SECTIONS.

REFERENCE: "Ada METHODOLOGIES: CONCEPTS AND REQUIREMENTS", (METHODMAN DOCT.) DoD Ada JOINT PROGRAM OFFICE, NOV. 1982.

SECTION 6

INTRODUCTION TO METHODS AND TOOLS

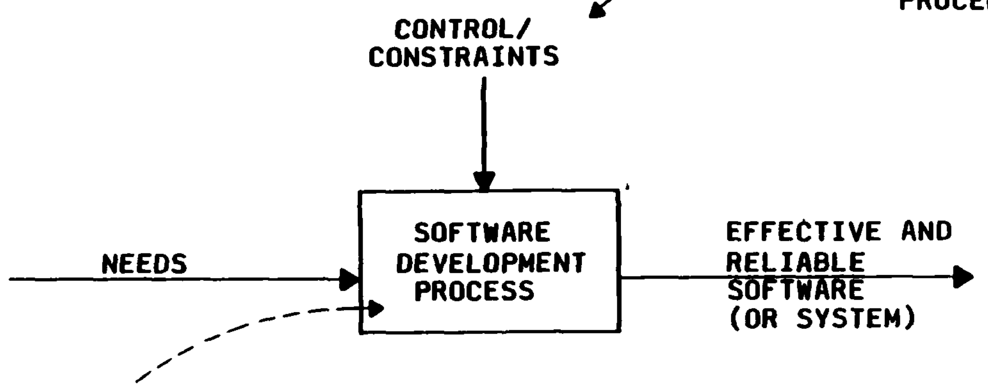
INSTRUCTOR NOTES

THIS PICTURE ILLUSTRATES VERY ABSTRACTLY WHAT THE SOFTWARE DEVELOPMENT PROCESS IS ALL ABOUT

"TAKING A SET OF NEEDS AND PRODUCING A SOFTWARE SYSTEM THAT MEETS THE NEED UNDER VARIOUS CONTROLS AND CONSTRAINTS

METHODOLOGIES GUIDE THE WAY THE PROCESS IS PERFORMED.

OUR SCOPE OF CONTROL



- MILITARY STANDARDS, PROCEDURES, BUDGETS
- METHODOLOGIES AND TOOLS ARE THE ONLY THINGS WE CAN AFFECT

INSTRUCTOR NOTES

GO OVER EACH POINT CAREFULLY. THIS SETS THE STAGE FOR THE STUDENTS. IT GIVES THEM ADDITIONAL WAYS TO LOOK AT THE METHODOLOGIES WE WILL BE DISCUSSING IN THIS COURSE.

ATTRIBUTES OF METHODOLOGIES

- A WELL-CONCEIVED METHODOLOGY HAS
  - CREATIVE ASPECTS
  - INTELLECTUAL ASPECTS
  - CLERICAL ASPECTS
  - MECHANICAL ASPECTS

- GOOD SOFTWARE ENGINEERING METHODOLOGIES SHOULD
  - IMPROVE EFFECTIVENESS AND PRODUCTIVITY OF SOFTWARE DEVELOPMENT ACTIVITIES
  - RESULT IN THE CREATION OF RELIABLE SOFTWARE
  - FIT TOGETHER TO FORM AN INTEGRATED SET OF METHODS
  - SEPARATE THE CREATIVE ASPECTS FROM THE MECHANICAL ASPECTS
  - PROMOTE AUTOMATION OF THE CLERICAL ASPECTS OF SOFTWARE DEVELOPMENT

INSTRUCTOR NOTES

MORE MOTIVATION. BE POSITIVE AND EMPHASIZE THE IMPORTANCE OF BOTH BULLETS.

WHY LEARN METHODOLOGIES?
(QUALITY VIEWPOINT)

- INCREASED EFFORT IN THE EARLIER ACTIVITIES OF A DEVELOPMENT WILL BE REFLECTED IN REDUCED COSTS FOR TESTING AND MAINTENANCE*
  - PREVENT ERRORS FROM BEING INTRODUCED
  - DETECT ANY ERRORS AT EARLIEST POSSIBLE TIME

- BY APPLYING A SET OF METHODOLOGIES YOU WILL ACHIEVE HIGHER QUALITY SOFTWARE THAT FULFILLS THE NEEDS

---

*REPAIR, ADAPTATION AND ENHANCEMENT THAT OCCURS AFTER INITIAL DEVELOPMENT.

INSTRUCTOR NOTES

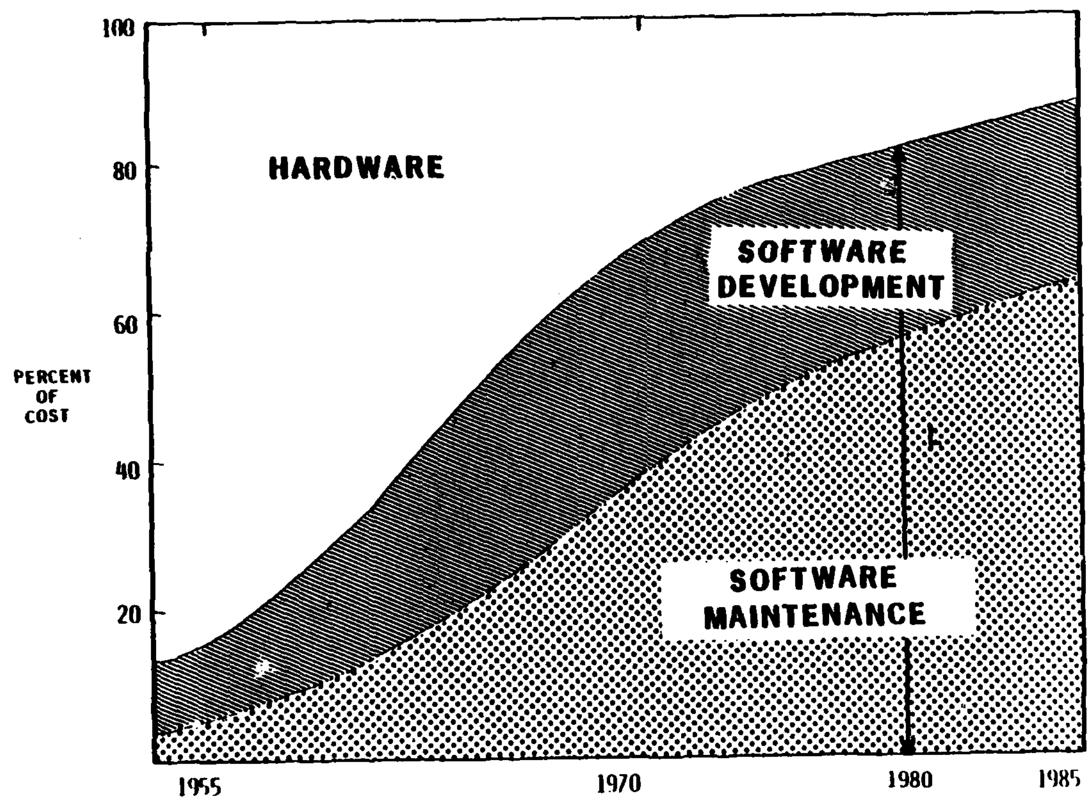
THIS SLIDE IS CLOSELY LINKED WITH THE PREVIOUS ONE. RE-EMPHASIZE THE IMPORTANCE OF METHODOLOGIES TO MINIMIZE MAINTENANCE COSTS IN THE FUTURE.

FIGURE OF 8.3% IN MUSA (SEE REFERENCES ON PAGE ii-i).

WHY LEARN METHODOLOGIES?
(COST VIEWPOINT)

- BY 1985 COMPUTER AND INFORMATION PROCESSING WILL REPRESENT 8.3% OF THE GNP

HARDWARE SOFTWARE COST TRENDS

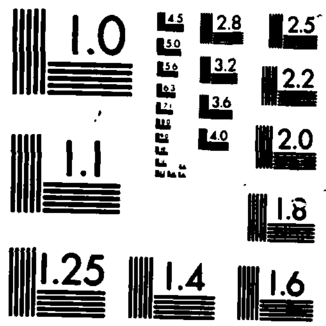


- IN 1980 L ≈ $40,000,000,000

AD-A165 123 ADA (TRADEMARK) TRAINING CURRICULUM: SOFTWARE ENGINEERING FOR MANAGERS M101 TEACHER'S GUIDE(U) SOFTECH INC WALTHAM MA 1986 DAAB07-83-C-K506

2/4

UNCLASSIFIED F/G 5/9 NL



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

INSTRUCTOR NOTES

TELL THE CLASS THAT THEY WILL SEE THE DETAILS OF THE DoD-STD-SDS REQUIREMENTS IN THE INTRODUCTORY SECTIONS TO ANALYSIS, DESIGN AND IMPLEMENTATION.

WHY LEARN METHODOLOGIES
(CONSTRAINT VIEWPOINT)

- FUTURE SOFTWARE DEVELOPMENTS WILL BE SO COMPLEX THAT WITHOUT A WELL-DEFINED SET OF METHODS THEY WILL BE IMPOSSIBLE TO ACCOMPLISH

- NEW MILITARY STANDARDS ARE REQUIRING THE USE OF A METHODOLOGY
  - DoD-STD-SDS

- THE POOL OF EXPERIENCED AND QUALIFIED SOFTWARE DEVELOPERS IS LIMITED
  - USE METHODOLOGIES TO IMPROVE PRODUCTIVITY

INSTRUCTOR NOTES

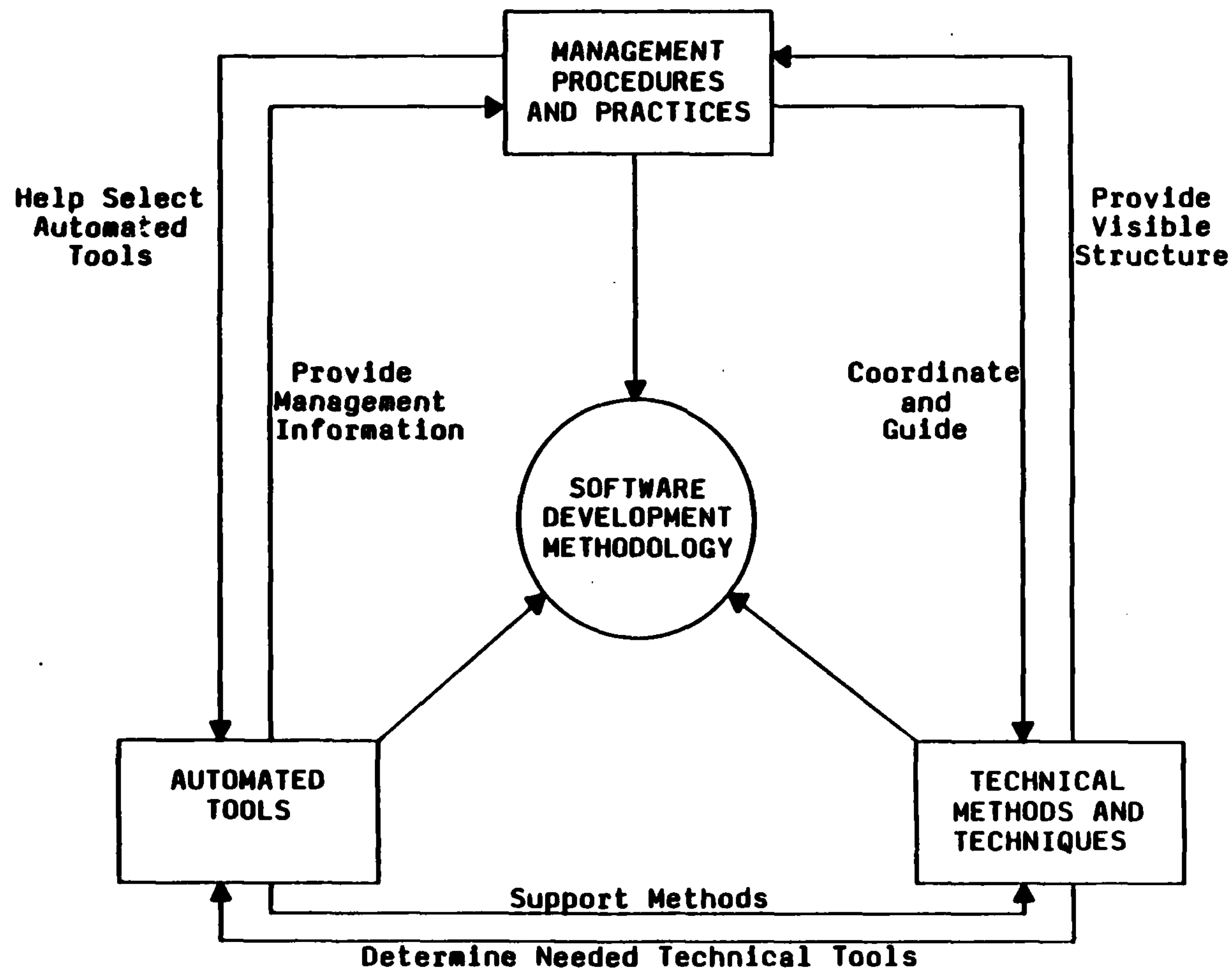
NOTE THAT ALL THREE ASPECTS MUST BE ADDRESSED IN A METHODOLOGY.

WALKTHROUGH THE DIAGRAM FROM METHODS, TO MANAGEMENT PRACTICES THEN AUTOMATED TOOLS.

EMPHASIZE THAT THIS COURSE FOCUSES ON THE TECHNICAL METHODS.

ASPECTS OF A METHODOLOGY

(IDEAL VIEW)



SOURCE: METHODMAN, 1982

VG 742.1

INSTRUCTOR NOTES

THE NEXT TWO SLIDES CHARACTERIZE AN IDEAL DEVELOPMENT METHODOLOGY WHICH WE COULD USE AS A MODEL TO MEASURE THE METHODOLOGIES WE WILL COVER.

WE USE THIS IN THE WRAP-UPS FOR ANALYSIS AND DESIGN TO COMPARE THE VARIOUS METHODS WE WILL DISCUSS.

REQUIREMENTS FOR A SOFTWARE DEVELOPMENT METHODOLOGY

(IDEAL VIEW)

- A METHODOLOGY SHOULD:
  - COVER THE <u>ENTIRE</u> DEVELOPMENT PROCESS, SIMPLIFYING TRANSITIONS BETWEEN PROJECT PHASES
  - ENHANCE COMMUNICATION AMONG ALL INTERESTED PERSONS AT ALL STAGES OF DEVELOPMENT
  - SUPPORT PROBLEM ANALYSIS AND UNDERSTANDING
  - SUPPORT BOTH TOP-DOWN AND BOTTOM-UP APPROACHES TO SOFTWARE DEVELOPMENT
  - SUPPORT SOFTWARE VALIDATION AND VERIFICATION THROUGH THE DEVELOPMENT PROCESS
  - FACILITATE THE CAPTURE OF DESIGN, IMPLEMENTATION, AND PERFORMANCE CONSTRAINTS

INSTRUCTOR NOTES

REQUIREMENTS FOR A SOFTWARE DEVELOPMENT METHODOLOGY

(IDEAL VIEW CONTINUED)

- A METHODOLOGY ALSO SHOULD:
  - SUPPORT THE SOFTWARE DEVELOPMENT ORGANIZATION
  - SUPPORT THE EVOLUTION OF A SYSTEM THROUGHOUT ITS EXISTENCE
  - BE SUPPORTED BY AUTOMATED AIDS
  - MAKE THE EVOLVING SOFTWARE PRODUCT VISIBLE AND CONTROLLABLE AT ALL STAGES OF DEVELOPMENT
  - BE TEACHABLE AND TRANSFERABLE
  - BE OPEN-ENDED

INSTRUCTOR NOTES

<u>THEME</u>: ANALYSIS IS CRITICAL IN THE DEVELOPMENT OF SOFTWARE, THUS WE MUST HAVE GOOD METHODS TO ADDRESS AND IDENTIFY THE <u>REAL</u> SOFTWARE REQUIREMENTS

<u>PURPOSE</u>: FOCUS THE STUDENTS' ATTENTION ON THE CRITICAL ASPECTS OF REQUIREMENTS ANALYSIS

<u>REFERENCES</u>:

1. FREEMAN, P., " A PERSPECTIVE ON REQUIREMENTS ANALYSIS AND SPECIFICATION" IBM DESIGN '79 SYMPOSIUM PROCEEDINGS; APRIL 1979

2. WEINBERG, G., "RETHINKING SYSTEMS ANALYSIS AND DESIGN" LITTLE, BROWN, MA; 1982

3. DoD-STD-SDS, PROPOSED MILITARY STANDARD FOR DEFENSE SYSTEM SOFTWARE DEVELOPMENT, DEC. 1983 (DoD-STD-1267)

SECTION 7

ANALYSIS INTRODUCTION

INSTRUCTOR NOTES

A REQUIREMENT IS A BINDING CONDITION WHICH STATES A MANDATORY CHARACTERISTIC OF AN ABSTRACT OR PHYSICAL OBJECT.

REQUIREMENTS MAY HAVE DIFFERENT FORMS: A SPECIFIC DESCRIPTION, A CONSTRAINT, AN EVALUATION CRITERIA FOR JUDGING QUALITY, OR IT MAY BE IMPLIED BY CONTEXT.

THERE'S ALWAYS MORE THAN ONE PARTY INVOLVED. ACHIEVING CONSENSUS IS AN IMPORTANT ASPECT OF REQUIREMENTS ANALYSIS. STRESS THAT REQUIREMENTS ANALYSIS IS A HUMAN ENDEAVOR.

ANALYSIS IMPLIES BOTH REQUIREMENTS ANALYSIS
AND SPECIFICATIONS

- A REQUIREMENT IS ...
  - AN EXPRESSION OF <u>NEED</u>
  - AN IMPOSED <u>DEMAND</u>
  - SOMETHING SOMEONE WILL PAY FOR
- REQUIREMENTS FORM A "CONTRACT BETWEEN USER AND DEVELOPERS"
- REQUIREMENTS ARE DOCUMENTED IN REQUIREMENTS SPECIFICATION(S)

INSTRUCTOR NOTES

THE ANALYSIS IS USUALLY CONDUCTED WHILE AT THE SPECIFICATION AND/OR DESIGN LEVEL

REQUIREMENTS ANALYSIS

- REQUIREMENTS ANALYSIS IS THE PROCESS BY WHICH THE FEASIBILITY OF REQUIREMENTS ARE DETERMINED, PRIOR TO THE DEVELOPMENT OF THE SYSTEM

- THE ANALYSIS IS TYPICALLY PERFORMED BY A SENIOR SYSTEMS EXPERT OR ANALYST WHO ESTABLISHES AND DOCUMENTS THE REQUIREMENTS

INSTRUCTOR NOTES

GOOD ANALYSTS CAN EXPRESS THE REAL REQUIREMENTS, AS OPPOSED TO THE STATED OR PERCEIVED ONES. IT IS A DIFFICULT JOB, AND MUST BE CAREFULLY THOUGHT OUT.

THE ANALYST IS THE <u>BRIDGE</u> BETWEEN
USERS AND DEVELOPERS

- GOOD ANALYSTS WILL:
  - POSTPONE DESIGN DECISIONS
  - PUT THEMSELVES IN THE USER'S CHAIR
  - QUESTION THE RATIONALE
  - UNDERSTAND THE <u>REAL</u> PROBLEM
  - CONSIDER SEVERAL SOLUTIONS
  - CLEARLY COMMUNICATE REQUIREMENTS AS WELL AS THE RESULTING IMPLEMENTATION

INSTRUCTOR NOTES

MANY SYSTEMS HAVE BEEN BUILT, TESTED AND MADE OPERATIONAL, ONLY TO BE NOT USED BECAUSE THEY DIDN'T MEET THE "REAL" REQUIREMENTS, ALTHOUGH THEY MET THE STATED REQUIREMENTS. REQUIREMENTS ARE HARD TO FORMULATE CORRECTLY!

CONSEQUENCES OF WRONG REQUIREMENTS

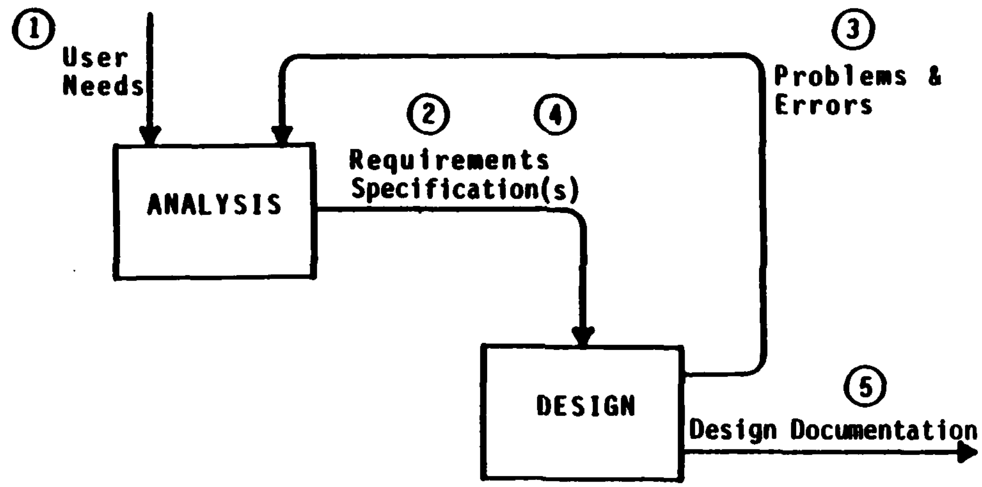
- WRONG REQUIREMENTS CAN CAUSE ...
  - SYSTEM REJECTION
  - SYSTEM PATCHING OR RETROFIT
  - INSTALLATION OF A DANGEROUS SYSTEM
  - FAILURE OF THE PROJECT
  - LOSS OF FUTURE BUSINESS

INSTRUCTOR NOTES

POINT OUT THAT OFTEN SOME LEVEL OF DESIGN IS NEEDED DURING ANALYSIS IN ORDER TO DETERMINE REQUIREMENTS FEASIBILITY.

ANALYSIS AND DESIGN ARE INTERRELATED

- **ANALYSIS AND DESIGN ITERATE ...**



INSTRUCTOR NOTES

POINT OUT THAT DESIGNERS ALSO CONTRIBUTE TO THE ANALYSIS PHASE. THEY CITE PROBLEMS AND ERRORS FROM <u>THEIR</u> VIEWPOINT OF <u>FEASIBILITY</u>, AND CONVEY THIS TO THE ANALYST. ALSO STATE THAT THERE MAY BE MULTIPLE REQUIREMENT SPECIFICATIONS (SOFTWARE, HARDWARE, SYSTEM, ETC.) THAT NEED TO BE CREATED AND ITERATED. CONFLICTS AMONG THEM NEED TO BE SPOTTED BY THE ANALYST.

DESIGNER'S ROLE IN ANALYSIS

- DURING ANALYSIS, DESIGNERS IDENTIFY ...
  - AMBIGUITIES
  - INCONSISTENCIES
  - INCOMPLETENESS
  - POTENTIAL TROUBLE AREAS
  - REQUIREMENTS HAVING DISPROPORTIONATE COST/SCHEDULE IMPACT

INSTRUCTOR NOTES

SECTION 8

ANALYSIS METHOD OVERVIEW

INSTRUCTOR NOTES

IN THIS SECTION WE WILL PRESENT AN OVERVIEW OF FIVE ANALYSIS TECHNIQUES. FOR EACH OF THESE TECHNIQUES WE WILL PRESENT EXAMPLES OF THE FORMAT(S) USED TO PRESENT INFORMATION WITH THE TECHNIQUE, WE WILL HIGHLIGHT IMPORTANT ASPECTS OF THE TECHNIQUE AND PROVIDE A CRITIQUE AS TO THE APPLICABILITY OF THE TECHNIQUE.

ANALYSIS METHOD OVERVIEW

- SADT - STRUCTURED ANALYSIS AND DESIGN TECHNIQUE
- SREM - SOFTWARE REQUIREMENTS ENGINEERING METHODOLOGY
- PSL/PSA - PROBLEM STATEMENT LANGUAGE/PROBLEM STATEMENT ANALYZER
- SSA - STRUCTURED SYSTEM ANALYSIS
- SCRP - SOFTWARE COST REDUCTION PROJECT

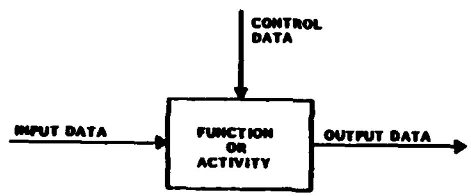
INSTRUCTOR NOTES

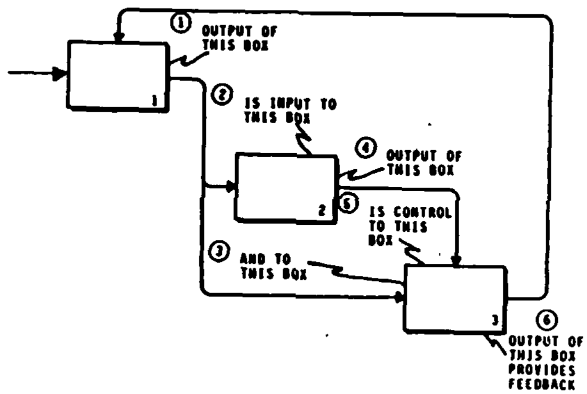
THE BASIC BUILDING BLOCKS OF SADT ARE BOXES AND ARROWS. BOXES ARE FUNCTIONS OR ACTIVITIES AND ARROWS REPRESENT DATA. DATA ENTERING ON THE LEFT SIDE IS AN INPUT THAT IS CONSUMED OR TRANSFORMED. DATA ENTERING ON THE TOP HAS SOME CONTROL OVER THE WAY IN WHICH THE ACTIVITY OR FUNCTION IS PERFORMED. DATA EXITING ON THE RIGHT IS PRODUCED OR MODIFIED BY THE FUNCTION.

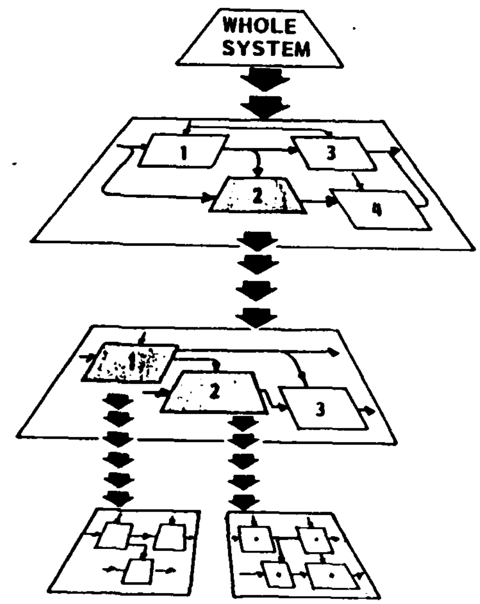
DIAGRAMS ARE COMPOSED OF THREE TO SIX BOXES INTERCONNECTED TO SHOW PRECEDENCE OR CONSTRAINT RELATIONSHIPS.

A MODEL IS A TOP-DOWN HIERARCHICAL DECOMPOSITION OF A SYSTEM STARTING FROM A SINGLE ACTIVITY (BOX) AND DECOMPOSING EACH BOX UNTIL ENOUGH DETAIL HAS BEEN PROVIDED TO SATISFY THE PURPOSE FOR WHICH THE MODEL WAS CREATED (USUALLY BEING ABLE TO ANSWER SOME SET OF QUESTIONS).

SADT







BASIC SYNTAX

BOXES AND ARROWS

SADT DIAGRAM

SADT MODEL

INSTRUCTOR NOTES

SADT CAN BE VIEWED AS A REFINEMENT AND FORMALISM OF THE TYPE OF BLOCK DIAGRAMMING THAT SEEMS TO BE SECOND NATURE TO MOST ENGINEERS. SADT ADDS SOME RULES AND CONVENTIONS THAT ALLOW LARGER AND MORE COMPLEX PROBLEMS TO BE ADDRESSED WITH MORE CONCISENESS THAN WOULD OTHERWISE BE POSSIBLE.

SADT WAS CREATED IN THE EARLY 70'S AND HAS BEEN USED ON HUNDREDS OF COMMERCIAL AND MILITARY APPLICATIONS.

SADT IS USUALLY TAUGHT IN A 5 OR 10 DAY COURSE AND HAS BEEN PICKED UP BY MANY PRACTITIONERS WITHOUT ANY FORMAL TRAINING.

SADT INCLUDES EXTENSIVE PROCEDURES FOR MANAGING THE FLOW OF INFORMATION, FOR CONDUCTING INTERVIEWS, AND FOR INSURING TIMELY AND THOROUGH REVIEW OF INFORMATION AS IT IS PRODUCED.

THESE PROCEDURES FACILITATE COMMUNICATION BY POTENTIALLY LARGE GROUPS OF ENGINEERS ON COMPLEX PROBLEMS - INSURES A CONSENSUS AT ALL LEVELS OF SYSTEM DEFINITION.

THE AIR FORCE OWNED VERSION IS KNOWN AS $IDEF_0$ ICAM (INTEGRATED COMPUTER AIDED MANUFACTURING) DEFINITION METHOD. TALK ABOUT BEHAVIOR MODELING AT YOUR OWN RISK.

SADT

- REPRESENTS A FORMALISM OF TRADITIONAL BLOCK DIAGRAMMING
- USED FOR OVER 10 YEARS ON A WIDE RANGE OF APPLICATIONS
- SIMPLE, LIMITED SYNTAX EASILY LEARNED AND USED
- INCLUDES MANAGEMENT AND REVIEW PROCEDURES
- SUPPORTS COMMUNICATION AND CONSENSUS BY LARGE DIVERSE GROUPS
- ALSO KNOWN AS $IDEF_0$
- PRIMARILY AN ANALYSIS TOOL BUT ALSO USED FOR ARCHITECTURAL DESIGN
- TECHNIQUE RECENTLY EXTENDED TO MODEL BEHAVIOR (STIMULUS/RESPONSE)

INSTRUCTOR NOTES

THE THREE IMPORTANT PRINCIPLES ON WHICH SADT ARE BUILT ARE STRUCTURING (STRICT TOP-DOWN HIERARCHY), ABSTRACTION, AND MODULARITY. ABSTRACTION IS SUPPORTED BY THE TOP-DOWN DECOMPOSITION WHICH FORCES ANY FUNCTION OR ACTIVITY TO INITIALLY BE DESCRIBED IN A SINGLE BOX. MODULARITY IS IMPOSED VIA THE DECOMPOSITION PROCESS WHICH ALWAYS CREATES 3 TO 6 NEW COMPONENTS WITH CLEARLY DEFINED INTERFACES. SADT IS LESS FORMAL THAN SOME TECHNIQUES, BUT MORE FORMAL THAN AD HOC BLOCK DIAGRAMMING TECHNIQUES OR FUNCTION AND CONTROL FLOW DIAGRAMS. SYNTAX RULES (I.E., CONTROL AND INPUT ARROWS) AND HIERARCHY RULES ARE EXAMPLES OF THESE FORMALISMS - MANY OTHERS EXIST.

THE PRIMARY GOAL OF SADT IS UNDERSTANDABILITY - REACHING A CONSENSUS AMONG A GROUP OF COMMUNICATING ANALYSTS. CORRECTNESS AND VERIFIABILITY OF THE SYSTEM DESCRIPTION ARE MUCH BETTER THAN WITH AD HOC DIAGRAMMING TECHNIQUES, BUT ARE NOT AS RIGOROUS AS SOME OTHERS.

THE HIERARCHICAL DECOMPOSITION RULES AID IN TRACING A REQUIREMENT THROUGH ALL LEVELS OF SYSTEM DESCRIPTION AIDING THE MAINTAINABILITY OF THE OVERALL SYSTEM.

SADT

PRINCIPLES

- STRUCTURING
  - TOP-DOWN HIERARCHICAL
- MODULARITY
  - DECOMPOSITION RULES
- ABSTRACTION
  - MULTIPLE LEVEL DESCRIPTION
- FORMALISM
  - REFINES INTUITIVE BLOCK DIAGRAMMING TECHNIQUE WITH HIERARCHY AND SYNTAX RULES

GOALS

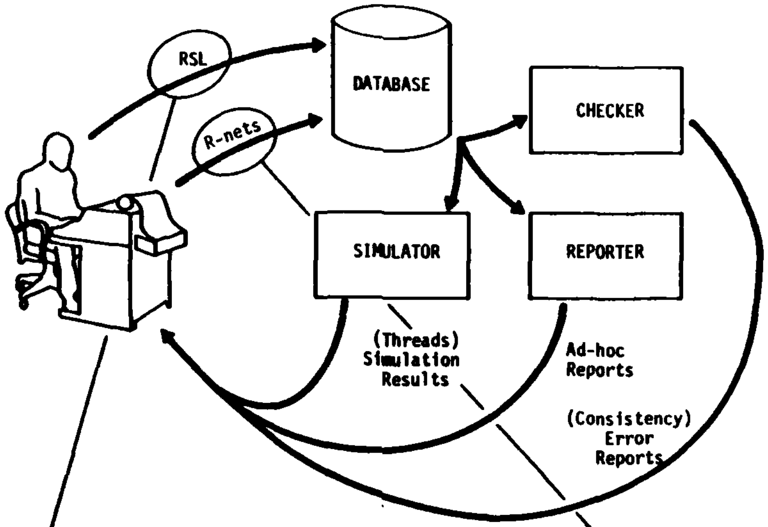
- UNDERSTANDABILITY
  - USES STRUCTURING, ABSTRACTION, AND MODULARITY TO AID COMMUNICATION
- CORRECTNESS, VERIFIABILITY
  - INTUITIVE NATURE OF DIAGRAMS AIDS REVIEW FOR CORRECTNESS
- MAINTAINABILITY, TRACEABILITY
  - HIERARCHICAL DECOMPOSITION ALLOWS EASY FOCUSING ON AREAS OF INTEREST

SREM BEGINS WITH THE TRANSLATION AND DECOMPOSITION OF SYSTEM LEVEL REQUIREMENTS. AS PART OF THE METHODOLOGY, A SET OF SOFTWARE SUPPORT TOOLS WERE IMPLEMENTED TO AUTOMATE MANY OF THE PREVIOUSLY MANUAL ACTIVITIES ASSOCIATED WITH REQUIREMENTS ENGINEERING. THESE SOFTWARE TOOLS FORM THE REQUIREMENTS ENGINEERING AND VALIDATION SYSTEM (REVS); REVS PROCESSING IS ACCOMPLISHED BY EXPRESSION OF THE SOFTWARE REQUIREMENTS IN THE STRUCTURED, FORMAL REQUIREMENTS SPECIFICATION LANGUAGE (RSL). A KEY CONCEPT OF REVS IS THAT ALL REQUIREMENTS ARE TRANSLATED INTO A CENTRAL DATABASE CALLED THE ABSTRACT SYSTEM SEMANTIC MODEL (ASSM). THE RSL STATEMENTS THEMSELVES ARE NOT STORED IN THE ASSM. INSTEAD, THEY ARE TRANSLATED INTO REPRESENTATIONS OF THE INFORMATION CONTENT OF THE REQUIREMENTS STATEMENTS. THIS PROVIDES AN EFFICIENT AND FLEXIBLE MEANS OF MAINTAINING A LARGE SOFTWARE SPECIFICATION IN A RELATIVELY SMALL COMPUTER DATABASE.

RSL EXPRESSES REQUIREMENTS IN AN UNAMBIGUOUS, MACHINE-PROCESSABLE LANGUAGE, AS OPPOSED TO FREE FORM (AND FREE CONTENT) ENGLISH. SUPPORT SOFTWARE IS AVAILABLE TO AUTOMATICALLY PROCESS THE REQUIREMENTS STATEMENTS AND PERFORM A WIDE RANGE OF NEEDED ACTIVITIES (E.G., IDENTIFY SYNTAX ERRORS USING THE RSL TRANSLATOR).

REQUIREMENTS ENGINEERING AND VALIDATION SYSTEM (REVS) CONSISTS OF A COMPUTER PROGRAM WITH A DATABASE. THE REQUIREMENTS INFORMATION WRITTEN FOR A SYSTEM WILL BE STORED IN THIS DATABASE AND A LIST OF ERRORS PROVIDED. THE DATABASE CAN BE CORRECTED UNTIL ALL DETECTED ERRORS ARE REMOVED. TOOLS ARE AVAILABLE TO MECHANICALLY GENERATE A SIMULATION FOR THE SYSTEM AND TO PROVIDE AUTOMATIC DOCUMENTATION.

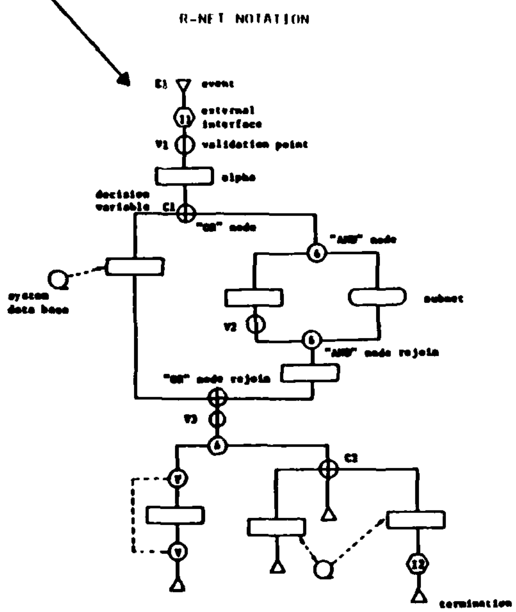
SREM



RSL

```
OUTPUT INTERFACE:  RADAR_ORDERS_BUFFER.
    ABBREVIATION BY:  ROB.
    CONNECTS TO:  RADAR
    RECEIVES FROM:  DPS.
    PASSES:  RADAR ORDER.
    ENTERED BY:  "MIKE RICHTER".

MESSAGE:  RADAR ORDER.
    MADE BY:  RADAR_COMMAND.

DATA:  STARTUP
    INCLUDES:  RO_ORDER_ID.
               STARTUP_TIME.

DATA:  SHUTDOWN
    INCLUDES:  RO_ORDER_ID.

DATA:  TRANSMIT_RECEIVE.
    INCLUDES:  RO_ORDER_ID
               RO_IMAGE_ID
               ALPHA_PHASE_TAPER
               BETA_PHASE_TAPER
               TRANSMIT_INFORMATION
               RECEIVE_INFORMATION
               NUMBER_OF_RANGE_GATES
               RANGE_GATE_INFORMATION
```

R-NET



INSTRUCTOR NOTES

INSTRUCTOR SHOULD READ ONE OF THE MANY MACK ALFORD PAPERS ON SUBJECT. SREM DEVELOPED INITIALLY AS AN AUTOMATED TOOL SPECIFICALLY FOR REAL TIME SYSTEMS. R-NETS PROVIDE A CIRCUIT-LIKE DESCRIPTION OF THE INPUT STIMULUS TO OUTPUT RESPONSE NATURE OF THE SYSTEM. RSL IS IN MANY WAYS SIMILAR TO PSL. SYSTEM AUTOMATICALLY GENERATES SIMULATION TEMPLATES FOR COMPONENTS THAT ARE THEN COMPLETED BY FOLLOWING IN PASCAL CODE.

HAS BEEN USED ON SEVERAL LARGE TRW PROJECTS.

SREM

- AUTOMATED REQUIREMENTS DEFINITION AND ANALYSIS SYSTEM
- DEVELOPED BY TRW FOR ARMY (BMD)
- MACK ALFORD, TRW, HUNTSVILLE IS POINT OF CONTACT
- TECHNIQUE DEVELOPED SPECIFICALLY FOR REAL TIME SYSTEMS
- HOSTED ON VAX/VMS BASED SYSTEM
- PROVIDES AUTOMATED CONSISTENCY CHECKING
- ASSISTS IN DEVELOPING SIMULATION
- RSL SIMILAR TO PSL
- R-NETS TRACE STIMULUS/RESPONSE NETWORK
- BEING EXTENDED TO SUPPORT DISTRIBUTED PROCESSING APPLICATIONS

INSTRUCTOR NOTES

SREM PROVIDES AN INTERESTING CONTRAST TO SADT. SREM IS MUCH MORE FORMAL AND IS AUTOMATED, THEREFORE IT IS MUCH BETTER AT ATTAINING CORRECTNESS AND VERIFIABILITY THAN SADT. WHAT IT GIVES UP IS UNDERSTANDABILITY BECAUSE IT ISN'T AS INTUITIVE AND DOESN'T SUPPORT ABSTRACTION IN THE SAME WAY.

IT DOES PROVIDE HOOKS FOR SIMULATION AND IS MORE EXPLICIT IN SHOWING INHERENT PARALLELISM. THESE PROVIDE MANY BENEFITS INCLUDING EFFICIENCY ANALYSIS.

SREM

PRINCIPLES

- FORMALISM
  - RSL, R-NETS
- MODULARITY
  - SUBNETS, RSL ENTITIES
- STRUCTURING
  - R-NET SYNTAX, RSL
- SEPARATION OF CONCERNS
  - RSL ENTITIES AND RELATIONSHIP
  - RSL/R-NET PARTITIONING
- UNIFORMITY
  - ENFORCED BY AUTOMATED CHECKING

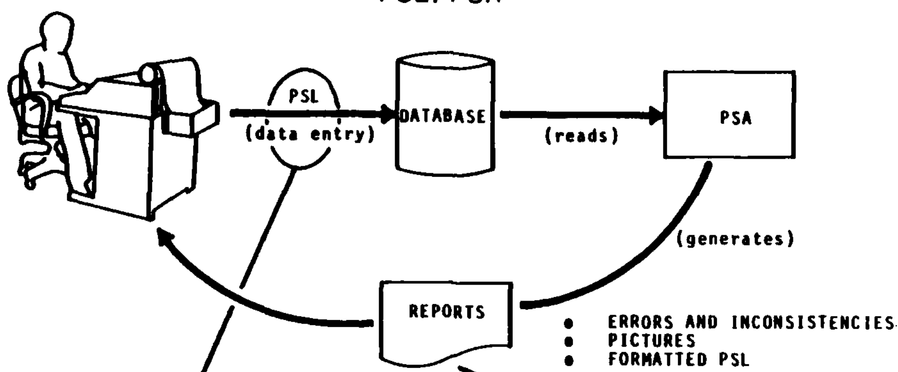
GOALS

- CORRECTNESS, VERIFIABILITY, TESTABILITY
  - SUPPORTED BY FORMAL LANGUAGE, CONSISTENCY CHECKING, DYNAMIC SIMULATION CAPABILITY
- MAINTAINABILITY, MODIFIABILITY
  - AIDED BY AUTOMATED DATABASE, EXTENSIVE REPORTS
- RELIABILITY
  - CONSISTENCY CHECKING AND SIMULATION
- EFFICIENCY
  - CAN BEGIN PARALLELISM AND TUNING ANALYSIS

INSTRUCTOR NOTES

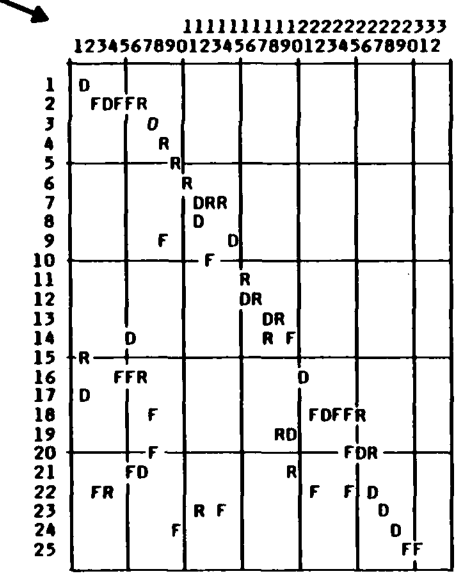
PSL/PSA IS ANOTHER AUTOMATED REQUIREMENTS ANALYSIS TECHNIQUE. THE INPUT - PROBLEM STATEMENT LANGUAGE IS ENTIRELY TEXTUAL. MANY TYPES OF REPORTS ARE GENERATED AS PART OF THE PROBLEM STATEMENT ANALYZER AND THE USER CAN ADD ADDITIONAL REPORTS (IN FORTRAN). MOST OF THESE REPORTS ARE TEXTUAL; HOWEVER SOME ARE GRAPHICAL.

PSL/PSA




```
PROCESS                  HOURLY-EMPLOYEE-PROCESSING:

   GENERATES           :  PAY-STATEMENT, ERROR-LISTING, HOURLY-EMPLOYEE-REPORT
   RECEIVES            :  TIME-CARD;
   SUBPARTS ARE        :  HOURLY-PAY-CHECK-VALIDATION, HOURLY-EMP-UPDATE,
                          HOURLY-REPORT-ENTRY-GENERATION,
                          HOURLY-PAYCHECK-PRODUCTION;
   PART OF             :  PAYROLL-PROCESSING;
   DERIVES             :  PAY-STATEMENT
      USING            :  TIME-CARD, HOURLY-EMPLOYEE-RECORD;
   DERIVES             :  HOURLY-EMPLOYEE-REPORT
      USING            :  TIME-CARD, HOURLY-EMPLOYEE-RECORD;
   DERIVES             :  ERROR-LISTING
      USING            :  TIME-CARD, HOURLY-EMPLOYEE-RECORD;
   PROCEDURE           :  1.  COMPUTE GROSS PAY FROM TIME CARD DATA.
                          2.  COMPUTE TAX FROM GROSS PAY.
                          3.  SUBTRACT TAX FROM GROSS PAY TO OBTAIN NET PAY.
                          4.  UPDATE HOURLY EMPLOYEE RECORD ACCORDINGLY.
                          5.  UPDATE DEPARTMENT RECORD ACCORDINGLY.
                          6.  GENERATE PAYCHECK.
                  NOTE:   IF STATUS CODE SPECIFIES THAT THE EMPLOYEE DID NOT WORK
                          THIS WEEK, NO PROCESSING WILL BE DONE FOR THIS EMPLOYEE;
   HAPPENS             :  NUMBER-OF-PAYMENTS TIMES-PER PAY-PERIOD;
   TRIGGERED BY        :  HOURLY-EMP-PROCESSING-EVENT;
   TERMINATION-CLAUSES:   NEW-EMPLOYEE-PROCESSING-EVENT;
   SECURITY IS         :  COMPANY-ONLY;
```

PSL



DATA PROCESS INTERACTION REPORT

INSTRUCTOR NOTES

- NOTE THAT PSL/PSA DOESN'T HELP YOU FORMALIZE THE REQUIREMENTS: IT'S JUST A TOOL.

- PSL/PSA IS BASED ON WHAT'S CALLED AN ENTITY/RELATIONSHIP MODEL. BASICALLY, RELATIONSHIPS (SPECIFIED AND IMPLIED) BETWEEN OBJECTS ARE DESCRIBED AND ARE RETRIEVABLE.

- THE TOOL IS CLAIMED TO BE METHODOLOGY INDEPENDENT. HOWEVER, BECAUSE OF THE RELATIONSHIPS AND OBJECTS ABLE TO BE DESCRIBED, AND THE PSL/PSA APPLICATION GUIDEBOOK, AN AD HOC METHODOLOGY EXISTS.

- PSL/PSA WAS ORIGINALLY DEVELOPED (CALLED URL/URA BY THE U.S. AIR FORCE) AS A DOCUMENTATION TOOL. OTHER USES OF THE TOOL FOLLOWED.

- PSL/PSA IS A DESCRIPTION VEHICLE. THAT'S WHERE ITS POWER LIES.

PSL/PSA

- PSL/PSA WAS DEVELOPED AS AN APPROACH TO IMPROVING SYSTEM/SOFTWARE DEVELOPMENT
- BASED ON AN ENTITY RELATIONSHIP MODEL
- BASED ON THREE PREMISES:
  - MORE EFFORT AND ATTENTION SHOULD BE DEVOTED TO "FRONT-END" PORTIONS OF THE DEVELOPMENT PROCESS
  - MAKE MAXIMUM USE OF AUTOMATION DUE TO LARGE AMOUNT OF INFORMATION THAT MUST BE HANDLED
  - PUT AUTOMATION IN CORRECT PERSPECTIVE - EMPHASIZE NEED FOR DOCUMENTATION
- USE OF PSL/PSA SIMILAR TO THAT OF SREM
- DEVELOPED BY DAN TEICHROEW AT THE UNIVERSITY OF MICHIGAN
- FAIRLY WIDELY USED ON LARGE SYSTEMS TO RECORD AND TRACK INTERFACES

INSTRUCTOR NOTES

WE SHOULD EXPECT THE PRINCIPLES AND GOALS FOR PSL/PSA TO BE QUITE SIMILAR TO SREM AND THEY ARE. SREM WITH ITS BUILT IN AIDS FOR SIMULATION AND ITS R-NETS HAS MORE SUPPORT FOR TESTABILITY AND GOES FURTHER THAN PSL IN ACTUALLY DESCRIBING FUNCTIONALITY AND BEHAVIOR.

COMPARED TO SADT, PSL CAN BE SEEN TO BE LESS INTUITIVELY UNDERSTANDABLE AND TO HAVE LESS SUPPORT FOR ABSTRACTION.

PSL/PSA

PRINCIPLES

- FORMALISM
  - PSL
- SEPARATION OF CONCERNS, STRUCTURING
  - PSL ENTITIES AND RELATIONSHIPS
- UNIFORMITY
  - PSA CONSISTENCY CHECKING REPORTS
- MODULARITY
  - SPECIFIC PSL RELATIONSHIPS (I.E., SUBPARTS)

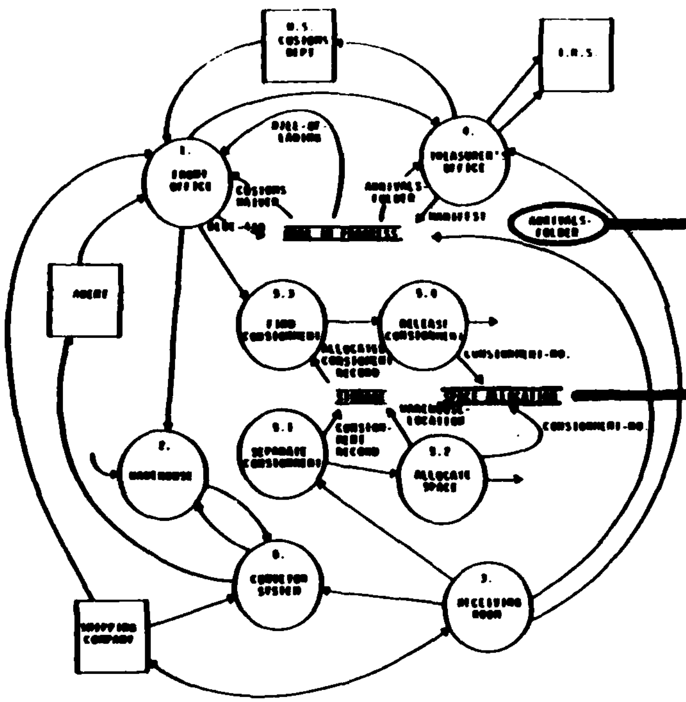
GOALS

- CORRECTNESS, VERIFIABILITY
  - PSA REPORTS
- MAINTAINABILITY, MODIFIABILITY
  - AIDED BY AUTOMATED DATABASE, PSA REPORTS
- RELIABILITY
  - CONSISTENCY CHECKING BY PSA

INSTRUCTOR NOTES

BY STRUCTURED SYSTEM ANALYSIS (SSA) WE ARE PRIMARILY TALKING ABOUT DATA FLOW DIAGRAMS (DFD) WHICH ARE ALSO KNOWN AS BUBBLE CHARTS OR DIAGRAMS. THE "BUBBLES" REPRESENT FUNCTIONS AND ACTIVITIES ARE LINKED BY ARROWS WHICH SHOW DATA FLOW. THE DATA IS MORE PRECISELY DEFINED BY A DATA DICTIONARY. POINT OUT THE DATA SINKS OR SOURCES (WORK IN PROGRESS AND SPACE ALLOCATION) AND THE EXTERNAL INTERFACES (IRS).

DATA FLOW DIAGRAM



DATA DICTIONARY ENTRIES

WORK-IN-PROGRESS = "FILE OF ALL INFO ABOUT RECEIVED GOODS"
= |SHIP-NAME + ARRIVAL-FOLDER + (MANIFEST)|

ARRIVAL FOLDER = |BILL-OF-LADING +
(ALLOCATION-TICKET) +
(CUSTOMS-WAIVER) + (FORM-448)|

BILL-OF-LADING = CONSIGNMENT-NO. +
DESTINATION +
SHIPPED-WEIGHT +
AGENT-NAME +
SHIP-NAME +
CONSIGNMENT-VALUE +
|PARCEL-NO. +
PARCEL-DESCRIPTION +
PARCEL-VALUE +
CUSTOMS-CODE|

ALLOCATION-TICKET = CONSIGNMENT-NO. +
WAREHOUSE-LOCATION +
RECEIVED-WEIGHT

CUSTOMS-WAIVER = CONSIGNMENT-NO. +
AUTHORIZATION-CODE

FORM-448 = CONSIGNMENT-NO. +
AGENT-NAME +
|PARCEL-NO. +
PARCEL-DESCRIPTION|

MANIFEST = SHIP-NAME + ARRIVAL-DATE +
|CONSIGNMENT-NO. + CONSIGNMENT-VALUE|

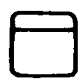
STORAGE-FILE = "FILE SHOWING HOW CONSIGNMENTS STORED"
= |CONSIGNMENT-NO. +
NUMBER-OF-SPACES-ALLOCATED +
|WAREHOUSE-LOCATION +
|PARCEL-NO.|||

SPACE-FILE = "FILE OF SPACE ALLOCATION IN WAREHOUSE"
= |WAREHOUSE-LOCATION +
CONSIGNMENT-NO.|

INSTRUCTOR NOTES

YOURDON DATA FLOW DIAGRAMS ARE ONE OF THE MOST WIDELY USED STRUCTURED ANALYSIS TECHNIQUES. THEY HAVE BEEN POPULARIZED THROUGH SEVERAL TEXTS BY TOM DEMARCO. THESE DFD DIAGRAMS ARE USED TO BUILD HIERARCHICAL MODELS MUCH LIKE SADT.

THE DATA DICTIONARY USES A LIMITED SET OF LOGICAL OPERATORS TO PRODUCE AN ACCURATE AND CONCISE DESCRIPTION OF DATA - THIS EMPHASIS REFLECTS THE WIDESPREAD USE OF THIS TECHNIQUE ON COMMERCIAL DATA PROCESSING APPLICATIONS. RECENT EXTENSIONS HAVE ATTEMPTED TO ADDRESS REAL TIME SYSTEMS BY INCLUDING STATE TRANSITION TABLES AND CONCEPTS SUCH AS EVENTS AND TRIGGERS.

ANOTHER SIMILAR TECHNIQUE HAS BEEN CREATED BY GANE AND SARSON AND USED "SQUARE" BUBBLES - RECTANGLES WITH ROUNDED CORNERS  AND A SIMILAR BUT LESS PRECISE DICTIONARY.

SSA

- DEVELOPED BY YOURDON AND DEMARCO - SEVERAL TEXTS AVAILABLE
- BUILD MODELS MUCH LIKE SADT
- DATA DICTIONARIES USE A SPECIFIC SET OF OPERATORS
  - = HIERARCHY (COMPRISES)
    + SEQUENCE (AND)
    [ ] SELECTION
    { } REPETITION
    ( ) OPTIONAL
- TECHNIQUE IS WIDELY USED IN COMMERCIAL APPLICATION
  - HAS BEEN EXTENDED FOR REAL TIME APPLICATIONS
- GANE AND SARSON HAVE SIMILAR TECHNIQUE WITH SLIGHTLY DIFFERENT GRAPHICS

INSTRUCTOR NOTES

WE WOULD EXPECT SSA TECHNIQUES TO BE VERY SIMILAR TO SADT - MANY PEOPLE CONSIDER THESE TWO TECHNIQUES TO BE INTERCHANGEABLE. HOWEVER THERE ARE DIFFERENCES AND THESE AFFECT THE RELATIVE IMPORTANCE OF PRINCIPLES AND GOALS. SADT HAS A MORE RIGOROUS MODULARITY CRITERIA - SSA DOESN'T HAVE ANY 3-6 BOX DECOMPOSITION RULES. SSA USES BUBBLES WITHOUT ANY DISTINCTION BETWEEN CONTROLS AND INPUTS SO IN THIS ASPECT SSA IS LESS FORMAL. HOWEVER, SSA'S DATA DICTIONARY IS AN ADDITIONAL FORMALISM OVER WHAT SADT OFFERS.

PRINCIPLES

- STRUCTURING
  - TOP-DOWN HIERARCHICAL
- ABSTRACTION
  - MULTIPLE LEVEL DECOMPOSITION
- MODULARITY
  - DECOMPOSITION RULES
- FORMALISM
  - SPECIALTY DATA DICTIONARY

GOALS

- UNDERSTANDABILITY
  - USES STRUCTURING AND ABSTRACTION TO AID UNDERSTANDABILITY
- CORRECTNESS, VERIFIABILITY
  - INTUITIVE NATURE OF DIAGRAMS AIDS REVIEW
- MAINTAINABILITY, TRACEABILITY
  - HIERARCHICAL DECOMPOSITION ALLOWS EASY FOCUSING ON AREAS OF INTEREST

INSTRUCTOR NOTES

THE IMPORTANT THINGS TO POINT OUT FOR THE SOFTWARE COST REDUCTION PROJECT IS THE SEGMENTATION OF THE REQUIREMENTS SPECIFICATION (A CLEAR APPLICATION OF SEPARATION OF CONCERNS) AND THE EXAMPLES OF TEMPLATES FOR FUNCTIONS, DATA ITEMS, AND MODES.

NOTE THAT THE INTERRELATIONSHIPS OF THESE TEMPLATES ARE NOT SHOWN EXPLICITLY BUT SHOULD BE POINTED OUT (I.E., FUNCTION TEMPLATE POINTS TO DATA ITEMS AND TO MODE CONDITION TABLE).

4.3.8 <u>Periodic function name</u>: Update Pullup Anticipation Cue Coordinates

<u>Modes in which function required:</u>

Weapon deliver: *Nattack*, *Noffset*, *BOC*, *BOCFlytoO*, *BOCoffset*, *CCIP*, *A/G Guns*, *Shrike*, *Walleye*, *Snattack*, *Snoffset*, *SBOC*, SBOCflytoO*, *SBOCoffset*

Test: *Grtest*

<u>Initiation and Termination Events:</u>
In the following table, !Range! refers to ground range to the !FLY-TO-point!

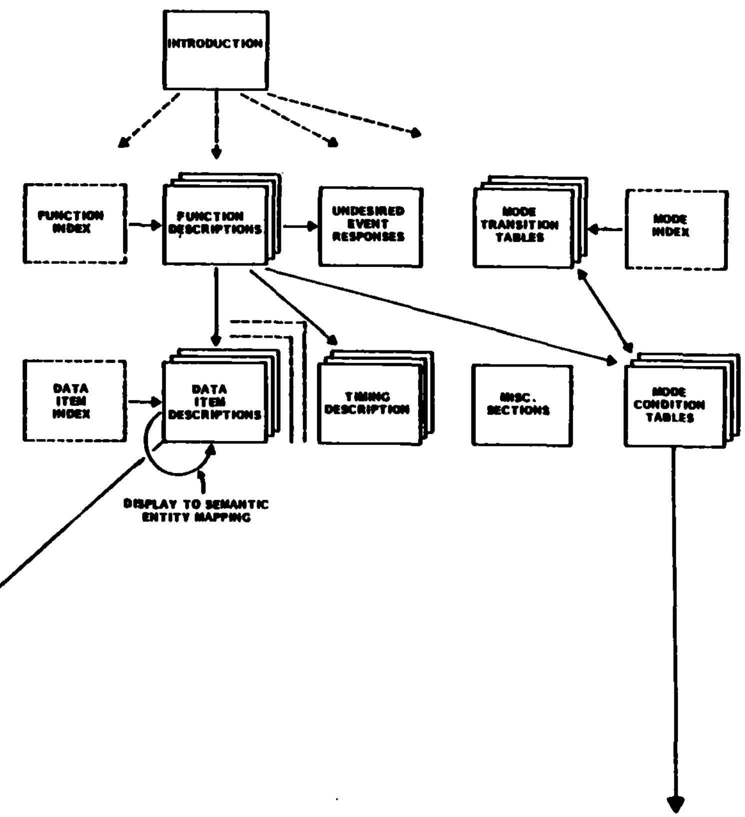
Event Table 4.3.8-a: When PUAC Updated (A)

MODES	EVENTS	
Nattack, *Noffset*, *BOCGlytoO*, *CCIP*, *A/G Guns*	@T (In mode)	X
*BOC, *BOCoffset*	@T(In mode AND !Range! lseq 30 nmi)	@T(!Range! gt 30 nmi)
Grtest	@F(!No WD MFS!)	@T(!No WD MFS!)
None of the listed modes	X	@T(!No WD MFS!)
ACTION	Initiation *PUAC in view*	Termination *PUAC out of view*

<u>Output Data Items</u>: //PUACAZ//, //PUACEL//

<u>Output description:</u>

In most modes, the PUAC shows the pilot how far he is from the "pullup point": *the point where he must execute a 4g pullup to avoid either the ground or the blast* radius of a released weapon.



INSTRUCTOR NOTES

INPUT DATA ITEM: MODE ROTARY SWITCH

ACRONYM: /MODEROT/

HARDWARE: TC-2 PANEL

DESCRIPTION: /MODEROT/ INDICATES THE SETTING OF THE MODE ROTARY SWITCH, A SIX POSITION ROTARY SWITCH ON THE TC-2 PANEL.

SWITCH NOMENCLATURE: PRES, POS, DEST, MARK, RNG/BRG, D-BHT, ALT-MSLP.

DATA TYPE: +ENUMERATION+

CHARACTERISTICS OF VALUES

VALUE ENCODING:

$None$	(000000),
$PRESPOS$	(100000),
$DEST$	(010000),
$MARK$	(001000),
RNG/BRG	(000100),
$DBHT$	(000010),
$ALTMSLP$	(000001)

INSTRUCTION SEQUENCE: READ 196 (CHANNEL 6)

DATA REPRESENTATION: TC-2 PANEL INPUT WORD 3, BIT 0-5

TIMING CHARACTERISTICS: /MODEROT/ = $None$ INDICATES THAT THE SWITCH IS IN TRANSITION BETWEEN TWO POSITIONS.

COMMENTS: THE MODE ROTARY SWITCH HAS GROWTH CAPABILITY TO EIGHT POSITIONS.

Condition / Mode	/IMSMODE/=	/ACAIRB/=	Align. stage completed	!latitude!	Other
DIG	$Norm$ OR $Gndal$	Yes	!CA stage!	ls 70o	!Doppler up! AND !IMS up!
DI	$Iner$ OR $Norm$ OR $Gndal$	Yes	!CL stage!	ls 80o	!Doppler used! AND !IMS up!
I	$Iner$ OR $Norm$ OR $Gndal$	X	!CL stage!	ls 80o	NOT !Doppler used! AND !IMS up!
Mag sl	$Magsl$	X	X	ls 80o	!IMS up!
Grid	$Grid$	X	X	X	!IMS up!
UDI	$Iner$	Yes	NOT !CL stage!	ls 80o	!IMS up! AND !Doppler used! AND !pitch small! AND !roll small!
OLB	$Iner$ OR $Norm$ OR $Gndal$ OR	X	NOT !CL stage!	ls 80o	X
IMS fail	X	X	X	X	!IMS down!
PolarDI	$Iner$ OR $Norm$ OR $Gndal$	Yes	!CL stage!	X	!Doppler used! AND !IMS up!
PolarI	$Iner$ OR $Norm$ OR $Gndal$	X	!CL stage!	X	NOT !Doppler used! AND !IMS up!

INSTRUCTOR NOTES

THE SCRP IS AN OUTGROWTH OF THE WORK DONE BY DAVID PARNAS. THIS PROJECT IS AN ATTEMPT TO APPLY SOFTWARE ENGINEERING PRINCIPLES TO A REAL, EXISTING SYSTEM - THE ONBOARD FLIGHT PROGRAM (OFP) OF THE A-7E. THIS IS THE ONLY FULL SCALE EXPERIMENT AIMED AT DEMONSTRATING THE FULL LIFE-CYCLE BENEFITS OF MODERN SOFTWARE ENGINEERING PRINCIPLES. THE PROJECT IS CURRENTLY IN THE INTEGRATION PHASE. SEPARATION OF CONCERNS AND BEING AS FORMAL AS POSSIBLE ARE THE TWO BASIC PRINCIPLES APPLIED TO THE REQUIREMENTS SPECIFICATION PHASE.

THE SEPARATION OF CONCERNS INCLUDES SEPARATING INPUT/OUTPUT MAPPING FROM FUNCTION DESCRIPTIONS AS WELL AS FUNCTIONALITY FROM BEHAVIOR, TIMING, AND ACCURACY.

A FUNDAMENTAL DIFFERENCE WITH THIS APPROACH IS THAT ANY OUTPUT OF THE SYSTEM IS ONLY CONTROLLED (OR SET) BY ONE FUNCTION - A 1 TO N RELATIONSHIP EXISTS BETWEEN FUNCTIONS AND OUTPUTS. THIS CREATES LOTS OF SMALL FUNCTIONS.

MODES ARE USED TO SIMPLIFY THE DESCRIPTION OF THE FUNCTIONS AND TO MAKE THE OVERALL COMPLEXITY OF THE SYSTEM EASIER TO COMPREHEND.

SCRP

- DEVELOPED BY NRL AND NWC, LED BY DAVID PARNAS
- TECHNIQUE BEING USED ON THE REDEVELOPMENT OF THE A-7E OFP
- BASED ON
  - SEPARATION OF CONCERNS
  - FORMALISM
  - ADDITIONAL DESIGN RELATED PRINCIPLES ADDRESSED LATER
- INPUTS AND OUTPUTS DESCRIBED INDEPENDENT OF FUNCTIONS
- 1 TO N RELATIONSHIP EXISTS BETWEEN FUNCTIONS AND OUTPUTS - VERY DIFFERENT FROM TRADITIONAL SPECS
- USES TEMPLATES AND TABLES FOR ADDITIONAL FORMALISM, SEPARATION OF CONCERNS
- USES PRECISELY DEFINED CONCEPTS OF MODES AND EVENTS
- MODES USED TO SIMPLIFY FUNCTION DESCRIPTIONS

INSTRUCTOR NOTES

SCRP IS VERY DIFFERENT FROM OTHER METHODS AND THEREFORE WE EXPECT TO SEE A DIFFERENT EMPHASIS ON GOALS AND UNDERLYING PRINCIPLES.

THE UNIQUE USE OF SEPARATION OF CONCERNS AND A DIFFERENT TYPE OF FORMALISM RESULT IN A HIGHLY MAINTAINABLE SPECIFICATION - THE IMPACT OF ANY CHANGE TO THE SYSTEM IS WELL CONTROLLED AND LOCALIZED.

UNDERSTANDABILITY IS ENHANCED BECAUSE THE SPEC IS PARTITIONED INTO MANY SMALL, EASILY GRASPED COMPONENTS; HOWEVER, THE "BIG PICTURE" IS NOT AS EASY TO GRASP AS WITH SADT, BUT A MUCH MORE DETAILED, PRECISE, CONCISE DESCRIPTION IS POSSIBLE WITH SCRP.

SADT AND SCRP CAN BE COMBINED TO GET THE BEST OF BOTH WORLDS - WRITE TO NEWPORT FOR DETAILS.

SCRP

PRINCIPLES

- SEPARATION OF CONCERNS
  - MAXIMIZED AT MULTIPLE LEVELS
- FORMALISM, UNIFORMITY
  - LIMITED SET OF PRIMITIVE CONSTRUCTS, HEAVY USE OF TABLES AND TEMPLATES
- MODULARITY, ABSTRACTION
  - PORTIONING INTO MODES AND SMALL FUNCTIONS
- STRUCTURING
  - ATTAINED THROUGH SEPARATION OF CONCERNS

GOALS

- MAINTAINABILITY, MODIFIABILITY
  - SEPARATION OF CONCERNS LIMITS IMPACT OF CHANGES
- CORRECTNESS, VERIFIABILITY, TESTABILITY
  - FORMALISM, STANDARD TABLES AND TEMPLATES FACILITATE REVIEW
- UNDERSTANDABILITY
  - SEPARATION OF CONCERNS PARTITIONS COMPLEXITY TO IMPROVE OVERALL UNDERSTANDABILITY
- PRODUCTIVITY
  - SEPARATION OF CONCERNS ALLOWS INEXPERIENCED ENGINEERS TO CONTRIBUTE EFFECTIVELY

INSTRUCTOR NOTES

THEME: DESIGN CONSISTS OF TWO SEPARATE SETS OF ACTIVITIES WHICH TAKE THE OUTPUT OF THE ANALYSIS (AND SPECIFICATION) PHASE OF THE LIFE CYCLE AND CREATES A "PAPER" MODEL OF THE SOFTWARE.

PURPOSE: TO PROVIDE INSIGHT INTO THE MAJOR ASPECTS OF THE DESIGN PHASE OF THE LIFE CYCLE.

REFERENCES: WEINBERG, G., "RETHINKING SYSTEMS ANALYSIS AND DESIGN" LITTLE, BROWN, MA; 1982

SECTION 9

DESIGN OVERVIEW

INSTRUCTOR NOTES

DESIGN IS A BLUEPRINT OF MODULES AND THEIR INTERCONNECTIONS. THE DESIGN PROCESS ALLOCATES THE FUNCTIONAL REQUIREMENTS TO A DESIGN STRUCTURE, PRESENTING THE FORM OR DESIGN STRUCTURE ALONG WITH SOME INDICATION OF WHERE THE VARIOUS FUNCTIONS ARE TO BE PERFORMED. THE STRUCTURE AND ALLOCATION SHOULD ALLOW THE PERFORMANCE AND ANALYTIC REQUIREMENTS TO BE FACTORED IN AND VERIFIED AS CONSTRAINTS.

THE INTERFACES BETWEEN THE COMPONENTS OF THE STRUCTURE ARE ALSO DEFINED AT THIS TIME. (MORE EMPHASIS IS PLACED ON THESE INTERFACES (E.G. PARNAS) THAN OTHERS (E.G. STRUCTURED DESIGN)).

THE ACTIVITY OF DESIGN IS A <u>MODELING</u> ACTIVITY. A DESIGN LEAVES CERTAIN DETAILS OUT UNTIL LATER. THERE IS A NEED TO BE ABLE TO COMPREHEND A LARGE AMOUNT OF THE SYSTEM TO BE SURE THAT THE GOALS OF THE PARTICULAR DESIGN STRATEGY (WHICH VARY) ARE BEING MET. DURING LATER STAGES OF THE DESIGN PROCESS DETAIL WILL BE ADDED AS WE EXAMINE SMALLER PARTS OF THE DESIGN.

THE DIFFERENT MODELING TECHNIQUES WE WILL COVER ENCOURAGE US TO LEAVE OUT DIFFERENT THINGS. LEAVING OUT THE DETAILS IS THE HARDEST PART OF THE DESIGN PROCESS.

WHAT IS DESIGN?

- DESIGN ...
  - TRANSLATES REQUIREMENTS SPECIFICATIONS INTO A BLUEPRINT OF THE SYSTEM
  - IS A MODEL OF THE SOFTWARE
  - IS DONE BY DESIGNERS

- FOR OUR PURPOSE DESIGN CONSISTS OF TWO MAJOR SUBPHASES
  - ARCHITECTURAL DESIGN (PRELIMINARY DESIGN)
  - DETAILED DESIGN

INSTRUCTOR NOTES

DESIGNERS DEAL WITH MORE OF THE SYSTEM AT ONE TIME THAN IMPLEMENTERS. THEY CONCENTRATE ON GLOBAL ISSUES, POSTPONING DECISIONS HAVING ONLY LOCAL SCOPE. THE DESIGNER PARTITIONS THE SYSTEM IN A MANNER THAT HIDES DECISIONS LIKELY TO CHANGE SEPARATELY INTO INDIVIDUAL MODULES.

CLEAN INTERFACES REALLY MEAN FULLY DEFINED INTERFACES. THE SIMPLER THESE INTERFACES ARE - FEWER OPTIONS, FEWER PARAMETERS - THE MORE LIKELY IT IS TO BE FOR THEM TO BE FULLY DEFINED. COMPLEX INTERFACES PROBABLY MEAN TOO MANY FUNCTIONS IN A SINGLE MODULE.

THE DESIGNER COMMUNICATES WITH THE ANALYST EITHER VERBALLY OR BETTER YET, THROUGH DOCUMENTATION TO MAP FUNCTIONAL REQUIREMENTS (AS WELL AS OTHERS SUCH AS RELIABILITY, MODIFIABILITY, ETC.). THE DESIGNER WORKS WITH IMPLEMENTERS TO ASSESS PERFORMANCE AND OTHER GOALS IN ORDER TO MAKE TRADE-OFFS AMONG THEM.

THE INTERFACE BETWEEN DESIGNERS AND ANALYSTS IS PROBABLY MORE DIFFICULT THAN BETWEEN DESIGNERS AND IMPLEMENTERS. DESIGNERS USUALLY WERE IMPLEMENTERS ONCE AND UNDERSTAND THE AREA WELL.

DESIGNER'S ROLE

- DESIGNERS ...
  - POSTPONE IMPLEMENTATION DECISIONS
  - HIDE THEIR DECISIONS INSIDE MODULES
  - WORRY ABOUT THE SOFTWARE'S STRUCTURE
  - SPECIFY ALGORITHMS AND CONTROL FLOW
  - MAKE CLEAN INTERFACES
  - COMMUNICATE WITH ANALYSTS AND IMPLEMENTERS

- THE DESIGNER IS THE BRIDGE BETWEEN ANALYSTS AND IMPLEMENTERS

INSTRUCTOR NOTES

SECTION 10

DESIGN METHOD OVERVIEW

INSTRUCTOR NOTES

THE FORMAT OF THIS DESIGN OVERVIEW PRESENTATION WILL BE THE SAME AS FOR THE ANALYSIS OVERVIEW - A SLIDE SHOWING GRAPHICS OR TEXTUAL FORMAT OF MATERIAL - A SLIDE PRESENTING SALIENT ASPECTS OF METHODOLOGY, AND A SLIDE CRITIQUING THE METHODOLOGY.

NOTE THAT WHILE ALL OF THESE METHODOLOGIES COVER SOME ASPECTS OF DESIGN, THEY ARE REALLY VERY DIFFERENT IN SCOPE AND EMPHASIS.

DESIGN METHOD OVERVIEW

SCRP - SOFTWARE COST REDUCTION PROJECT

OBJECT ORIENTED

SD - STRUCTURED DESIGN

JACKSON/WARNIER-ORR

HOS - HIGHER ORDER SOFTWARE

INSTRUCTOR NOTES

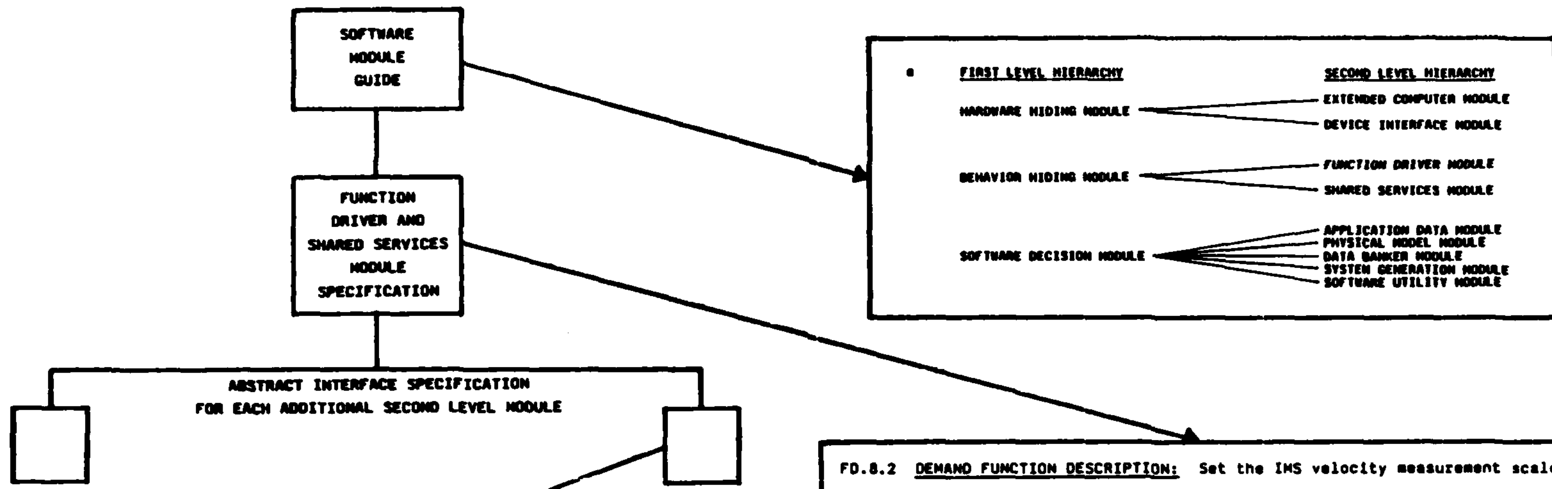
THE SCRP METHODOLOGY IS NOT JUST BASED ON A SET OF PRINCIPLES - INFORMATION HIDING, SEPARATION OF CONCERNS, ABSTRACT INTERFACES - BUT ALSO PROVIDES AN EXAMPLE FOR HOW TO PROCEED THROUGH THE DESIGN PROCESS STEP-BY-STEP AND HOW TO ORGANIZE THE DOCUMENTATION OF THE DESIGN.

THE FIRST STEP OF THE DESIGN - THE HIGH LEVEL MODULAR DECOMPOSITION - IS DOCUMENTED IN THE SOFTWARE MODULE GUIDE. THIS IS THE ONLY DESIGN DOCUMENT (25 PAGES) THAT ALL IMPLEMENTORS OR MAINTAINERS MUST READ.

MOST OF THE PROCESS OR TASKING STRUCTURE OF THE SYSTEM IS DOCUMENTED IN THE FUNCTION DRIVER AND SHARED SERVICES MODULE SPECIFICATION. THIS IS WHERE THE STIMULUS/RESPONSE BEHAVIOR OF THE SYSTEM IS DOCUMENTED.

THE ABSTRACT INTERFACE SPECIFICATIONS (THERE ARE SEVERAL) DESCRIBE THE INTERFACE TO THE COMPUTER AND ALL THE PERIPHERAL DEVICES AND TO CERTAIN OTHER MODULES WHICH IMPLEMENT SOFTWARE DECISIONS (I.E., THE DATABASE MODULE) WHICH ARE NOT DERIVED FROM THE FUNCTIONAL SPECIFICATION.

SCRP



DI.15. **SWITCH BANK INTERFACE**

DI.15.1. Introduction

The switch bank is a data entry device consisting of a set of switches. The switches fall into two classes, depending on the *number of states*: toggles with two states and selectors with more than two states. The switches and switch positions are named in accordance with the nomenclature in reference (1).

DI.15.2. Interface Overview

DI.15.2.1 Access Function Table

Function name	Parm type	Parm info	Undesired events
+G_AUTOCAL_TOGGLE+	p1:logical;O	!+Auto-cal sw+1!	None
+G_FLY_TO_NUM_SELECTOR+	p1:integer;O	!+Fly to num+!	
+G_FLY_TO_SELECTOR+	*p1:fly_to_state;O*	*!+Fly to state+!*	
+G_MAP_DECENTER_TOGGLE+	p1:logical;O	!+Map decenter+!	
+G_MAP_HOLD_TOGGLE+	p1:logical;O	!+Map hold+!	
+G_MAP_LDG_TOGGLE+	p1:logical;O	!+Map ldg+!	
+G_MAP_NORTH_UP_TOGGLE+	p1:logical;O	!+Map north-up+!	
+G_MAP_SCALE_TOGGLE+	p1:logical;O	!+Map scale sw+!	
+G_PANEL_MODE_SELECTOR+	p1:panel_mode;O	!+Panel mode+!	
+G_PANEL_UPDATE_SELECTOR+	p1:update;O	!+Update+!	
+G_PRES_POSITION_SELECTOR+	p1:pp_mode;O	!+Pres pos+!	
+G_SELF_TEST_TOGGLE+	p1:logical;O	!+Self-test+!	

FD.8.2 DEMAND FUNCTION DESCRIPTION: Set the IMS velocity measurement scale.

Mnemonic: +FD_IMS_SCALE_D+

Output produced:

Item	Type	Access program
IMS scale	Imsscale	+DI_S_IMS_SCALE+

Function definition:

Event Table FD.8.2-a -- Changing the IMS Velocity Measurement Scale

MODES	EVENTS	
Landaln *Lautocal* *Ol Update*	@T(In mode)	X
HUDaln	@T(In mode) WHEN (!_IMS mode_! = $Gndal$)	@T(In mode) WHEN(!+IMS mode+! = ($Norm$) OR $Iner$))
Airaln *Sautocal* *SINSaln* *DI *DIG *I* *OLB* *PolarDI* *PolarI* *UDI*	X	@T(In mode)
Output value:	$Fine$	$Coarse$

INSTRUCTOR NOTES

NOTE - READ SCRP PAPERS ON MODULE GUIDE AND ABSTRACT INTERFACES.

THIS PROJECT HAS SUCCESSFULLY APPLIED THE PRINCIPLE OF INFORMATION HIDING TO A COMPLEX SYSTEM AND HAS DESIGNED AND DOCUMENTED THE SYSTEM USING THIS APPROACH. ALL DOCUMENTS ARE READILY AVAILABLE AND SHOW HOW PRINCIPLE WAS APPLIED.

THIS APPROACH REQUIRES THE ANALYST AND DESIGNER TO PREDICT WHAT REQUIREMENTS ARE LIKELY TO CHANGE AND THEN TO PARTITION THE MODULES OF THE SYSTEM SO THAT THE EFFECT OF THESE CHANGES IS LIMITED. (I.E., IF WE EXPECT A PARTICULAR SENSOR TO POSSIBLY BE REPLACED, THEN WE DESIGN A MODULE WHICH PROVIDES AN INTERFACE TO THE REST OF THE SYSTEM TO PROVIDE THE ESSENTIAL INFORMATION OF THE SENSOR IN A MANNER THAT WON'T BE CHANGED IF WE CHANGE THE SPECIFIC SENSOR).

BECAUSE OF THE SEPARATION OF CONCERNS HAS BEEN APPLIED TO BOTH THE FUNCTIONAL REQUIREMENTS DESCRIPTION AND THE DESIGN DESCRIPTION IT IS POSSIBLE TO BUILD IN AN EXTREMELY HIGH DEGREE OF TRACEABILITY FROM REQUIREMENTS TO DESIGN.

THE HIGHER LEVEL OF THE DESIGN DECOMPOSITION ARE APPLICABLE TO A WIDE CLASS OF EMBEDDED SYSTEMS. DAVE PARNAS HAS BET HIS RIGHT ARM THAT THE FIRST LEVEL DECOMPOSITION IS UNIVERSALLY APPLICABLE AND HIS LEFT TOE THAT THE SECOND LEVEL IS UNIVERSALLY REUSABLE.

SCRP

- MODULAR DECOMPOSITION BASED ON <u>INFORMATION HIDING</u>
- INDEPENDENT MODULES HIDE DECISIONS THAT ARE LIKELY TO CHANGE INDEPENDENTLY
- MODULES ARE DESCRIBED USING STANDARD ABSTRACT INTERFACE TEMPLATE
  - ABSTRACT BECAUSE IT ONLY SHOWS THE INTERFACE TO THE MODULE THAT WILL NOT CHANGE IF THE HIDDEN DECISION (INFORMATION) CHANGES
- EXTREMELY HIGH DEGREE OF TRACEABILITY FROM FUNCTIONAL REQUIREMENTS TO DESIGN DESCRIPTION
- HIGH LEVEL DESIGN DECOMPOSITION, FIRST TWO LEVELS; CONSIDERED TO BE REUSABLE FOR ALL EMBEDDED APPLICATIONS

INSTRUCTOR NOTES

THE SCRP WAS INSTIGATED TO DEMONSTRATE THAT THE USE OF ABSTRACT MODULE INTERFACES, WHERE THE MODULARIZATION CRITERIA IS BASED ON INFORMATION HIDING, WOULD REDUCE LIFE-CYCLE MAINTENANCE COSTS - DATA HAS ALREADY BEEN COLLECTED TO VERIFY THIS. THUS THE IMPORTANCE OF THESE PRINCIPLES AND GOALS.

AN ANALYSIS OF THE SCRP DOCUMENTATION WILL SHOW THIS MODULARIZATION APPROACH AND THE USE OF SEPARATION OF CONCERNS PROVIDES FOR AN EXTREMELY HIGH DEGREE OF TRACEABILITY FROM REQUIREMENTS TO DESIGN AND A DESIGN THAT IS EASILY UNDERSTOOD AND ONE IN WHICH ANY DESIGNER CAN QUICKLY LOCATE HIS/HER AREA OF CONCERN.

THE FORMALISM AND UNIFORMITY MAKE THE DESIGN REVIEWABLE FOR CORRECTNESS.

AGAIN THE HIGH LEVEL OF MODULARITY AND THE SEPARATION OF CONCERNS ALLOWS FOR MANY TASKS WHICH CAN BE PURSUED IN PARALLEL, SOME BY RELATIVELY JUNIOR ENGINEERS, AND THEN QUICKLY REVIEWED BY MORE EXPERIENCED ENGINEERS.

SCRP

PRINCIPLES

- ABSTRACTION, MODULARITY, HIDING
  - SYSTEM PARTITIONED INTO HUNDREDS OF MODULES USING INFORMATION HIDING AND ABSTRACT INTERFACES
- SEPARATION OF CONCERNS
  - USED AT SEVERAL LEVELS
- UNIFORMITY, FORMALISM
  - LIMITED SET OF PRIMITIVE CONSTRUCTS, HEAVY USE OF TABLES AND TEMPLATES
- STRUCTURING
  - ATTAINED THROUGH MODULARIZATION STRATEGY

GOALS

- MAINTAINABILITY, MODIFIABILITY
  - MODULARIZATION PRINCIPLES MINIMIZE IMPACT OF CHANGE
- TRACEABILITY, UNDERSTANDABILITY
  - SEPARATION OF CONCERNS, MODULAR STRUCTURE FACILITATE TRACING FROM REQUIREMENTS TO DESIGN
- CORRECTNESS, VERIFIABILITY, TESTABILITY
  - FORMALISM, STANDARD TABLES AND TEMPLATES FACILITATE REVIEWS
- PRODUCTIVITY
  - SEPARATION OF CONCERNS, ABSTRACT INTERFACES, AND MODULARITY ALLOW FOR PARALLEL DEVELOPMENT BY ENGINEERS AT VARIED LEVELS OF EXPERIENCE

INSTRUCTOR NOTES

OBJECT ORIENTED DESIGN IS A STRATEGY, PRIMARILY AIMED AT THE MORE DETAILED LEVELS OF DESIGN (ALTHOUGH PROPONENTS WON'T NECESSARILY AGREE).

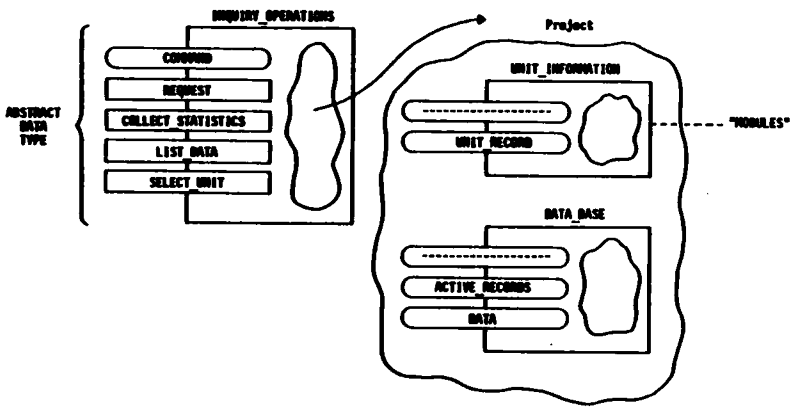
THE METHOD CONSISTS OF THE FOUR STEPS IDENTIFIED. DEVELOPING ON INFORMAL STRATEGY IS USUALLY ACCOMPLISHED BY PRODUCING A WRITTEN DESCRIPTION. IDENTIFYING DATA OBJECTS IS ACCOMPLISHED BY UNDERLINING THE NOUNS. IDENTIFYING THE OPERATIONS IS ACCOMPLISHED BY UNDERLINING VERBS. ESTABLISHING INTERFACES IS MORE COMPLEX.

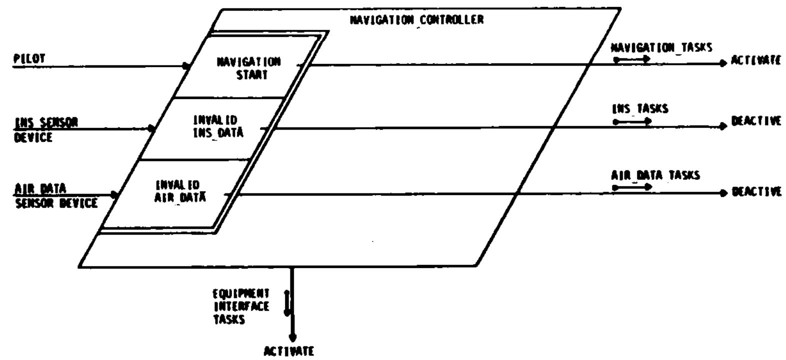
EXAMPLES OF THE OUTPUT ARE Ada CODE FRAGMENTS AND VARIOUS GRAPHICS THAT HAVE BEEN DEVELOPED BY VARIOUS AUTHORS TO USE WITH THIS APPROACH.

OBJECT ORIENTED

- METHOD
  - DEVELOP AN INFORMAL STRATEGY
  - IDENTIFY DATA OBJECTS
  - IDENTIFY OPERATIONS ON THESE OBJECTS
  - ESTABLISH INTERFACES

```
with PROJECT;
use PROJECT;
package INQUIRY_OPERATIONS is
   type COMMAND is (COLLECT_STATISTICS, LIST_DATA, QUIT, SELECT_UNIT);
   function REQUEST return COMMAND;
   procedure COLLECT_STATISTICS;
   procedure LIST_DATA;
   procedure SELECT_UNIT;
end INQUIRY_OPERATIONS;
```





INSTRUCTOR NOTES

THIS TECHNIQUE WAS FORMALIZED BY GRADY BOOCH IN HIS BOOK "SOFTWARE ENGINEERING IN Ada." THERE ARE SEVERAL PAPERS BY HIM AND OTHERS ON THIS TOPIC.

ABSTRACT DATA TYPES - SET OF VALUES, DATA STRUCTURES AND THE ASSOCIATED OPERATIONS RELATED TO THAT DATA TYPE.

OBJECT ORIENTED DESIGN OVERVIEW

- EVOLVED FROM WORK DONE BY G. BOOCH AND INTEL CORPORATION ON METHODS TO EXPRESS Ada SOFTWARE MODULE DESIGNS

- BASED ON THE SAME CONCEPTS AS SCRP DESIGN METHODS
  - ABSTRACTION
  - MODULARITY
  - HIDING

- DIFFERS FROM SCRP DESIGN METHOD IN ...
  - EMPHASIS ON ABSTRACTION OF DATA (ABSTRACT DATA TYPES)
  - CRITERIA FOR MODULARIZATION BASED ON DATA TYPE AND OPERATIONS
  - BOTTOM-UP VS. TOP-DOWN VIEW OF DESIGN

INSTRUCTOR NOTES

DIFFERENCES IN CRITERIA FOR MODULARITY (COMPARED TO SCRP) SUPPORT DIFFERENT GOALS. SCRP ATTEMPTS TO IDENTIFY LIKELY CHANGES TO THIS SYSTEM AND BUILDS MODULES TO HIDE EFFECTS. OBJECT ORIENTED USES A MORE GENERALIZED APPROACH AND BUILDS MODULES AROUND DATA STRUCTURES. THIS IS MORE LIKELY TO BUILD MODULES USABLE ON A DIFFERENT SYSTEM, BUT MAY NOT ACCOMMODATE CHANGES TO THIS SPECIFIC SYSTEM AS EASILY.

REUSABILITY AND TRANSPORTABILITY DON'T CONFLICT WITH MAINTAINABILITY AND MODIFIABILITY - THEY REFLECT A CHANGE IN EMPHASIS.

OBJECT ORIENTED

PRINCIPLES

- ABSTRACTION, MODULARITY, HIDING
  - SYSTEM PARTITIONED AROUND DATA OBJECTS - USES Ada PACKAGE SPECS FOR ABSTRACT INTERFACE
- UNIFORMITY, FORMALISM
  - INHERENT IN THE USE OF Ada SYNTAX AND SEMANTICS
- STRUCTURING
  - USES Ada'S STRUCTURING CONSTRUCTS

GOALS

- REUSABILITY, TRANSPORTABILITY
  - ATTEMPTS TO BUILD GENERIC REUSABLE COMPONENTS
- CORRECTNESS, VERIFIABILITY, TESTABILITY
  - *USE OF* Ada SYNTAX ALLOWS FOR COMPILER AND TOOL PROCESSING AND CONSISTENCY CHECKING

INSTRUCTOR NOTES

STRUCTURED DESIGN MEANS DIFFERENT THINGS TO DIFFERENT PEOPLE. WE ARE TALKING ABOUT SD AS POPULARIZED BY YOURDON AND CONSTANTINE.

BASIC STRUCTURES ARE DATA FLOW GRAPHS - MUCH LIKE DATA FLOW DIAGRAMS USED IN SSA, BUT USED TO DESCRIBE THE IMPLEMENTATION FLOW OF DATA (RATHER THAN THE EXTERNAL VIEW REFLECTED BY DFDs).

THESE DATA FLOW GRAPHS ARE ANALYZED TO PRODUCE STRUCTURE CHARTS WHICH REFLECT THE HIERARCHICAL MODULAR STRUCTURE.

INDIVIDUAL MODULES ARE THEN DESIGNED IN DETAIL BY MINI-SPECS - USUALLY A PDL TAILORED TO THE IMPLEMENTATION LANGUAGE.

STRUCTURED DESIGN

- THE BASIC SYNTAX, USED MOST OFTEN COMPRISES FOUR SYMBOLS ...

MODULE	□	(SOFTWARE FUNCTION)
INTERFACE	|	(THE PLACE WHERE DATA IS PASSED BETWEEN MODULES)
DATA	o→	(APPLICATION)
	•→	(CONTROL)

BUBBLE CHART SYNTAX

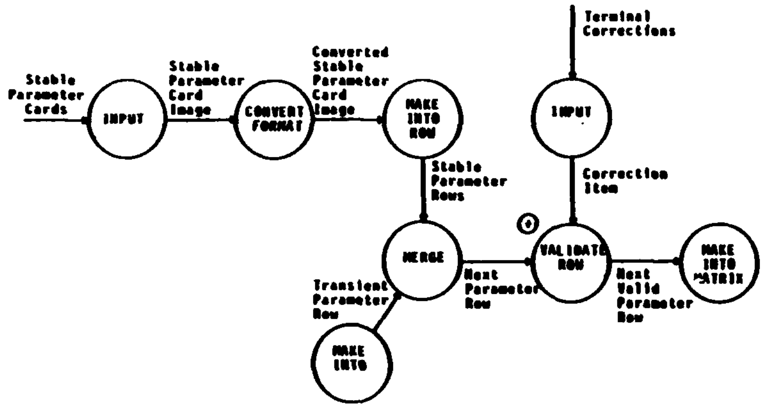
PIPELINE	→	(DATA)
TRANSFORM	○	(FUNCTION)

INSTRUCTOR NOTES

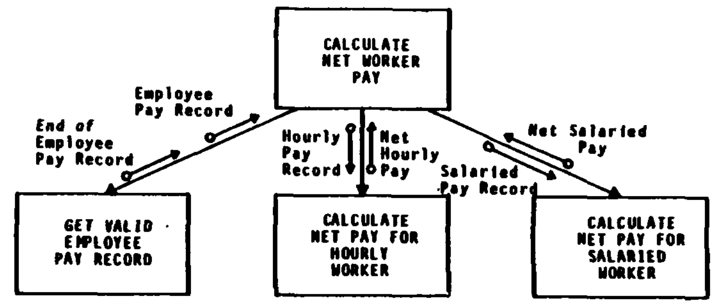
STRUCTURED DESIGN (Continued)

DATA FLOW GRAPH

- ARROWS CONNECT BUBBLES TOGETHER INTO A NETWORK:



- THE TOTAL EFFECT IS A HIERARCHY OF MODULES ...



INSTRUCTOR NOTES

STRUCTURED DESIGN DOES OFFER A "COOKBOOK" APPROACH FOR PROCEEDING FROM A DATA FLOW ANALYSIS OF THE PROPOSED IMPLEMENTATION TO A MODULAR STRUCTURE. TWO DIFFERENT STRATEGIES ARE AVAILABLE; TRANSFORM CENTERED ATTEMPTS TO FIND THE MOST IMPORTANT TRANSFORM THAT IS PERFORMED ON THE DATA AND BUILD THE MODULAR STRUCTURE AROUND THIS TRANSFORM (AS WE PROGRESS DOWN THE HIERARCHY WE USE THE NAME STRATEGY LOOKING FOR THE NEXT MOST IMPORTANT TRANSFORMS).

TRANSACTION CENTERED DESIGN LOOKS FOR THE BUBBLE WITH THE HIGHEST VOLUME OF TRANSACTIONS (THE RECEIVE AND DISPATCH CENTER OF THE SYSTEM) AND STRUCTURES THE MODULES AROUND IT. THESE TWO STRATEGIES ARE NOT NECESSARILY IN CONFLICT.

ADDITIONAL METRICS ARE USED TO REVIEW THE QUALITY OF THE PROPOSED DESIGN.

COUPLING MEASURES THE RELATIONSHIP BETWEEN MODULES - WE WANT TO MINIMIZE COUPLING.

COHESION MEASURES THE RELATEDNESS OF PIECES WITHIN A MODULE - WE WANT TO MAXIMIZE COHESION.

STRUCTURED DESIGN

- DATA FLOW GRAPHS ARE DERIVED FROM SOFTWARE REQUIREMENTS BY A DECOMPOSITION PROCESS

- DATA FLOW GRAPHS ARE NETWORKS OF TRANSFORMS CORRECTED BY PIPELINES OF CONTINUOUS DATA STREAMS

- STRUCTURE CHARTS ARE DERIVED USING ONE OF TWO STRATEGIES
  - TRANSFORM CENTERED - THE MAJOR TRANSFORMATION IS MADE AT THE TOP OF THE STRUCTURE HIERARCHY
  - TRANSACTION CENTERED - THE BUBBLE WITH THE MOST INPUTS AND OUTPUTS IS MADE AT THE TOP OF THE STRUCTURE
  - VERY OFTEN BOTH CONDITIONS ARE SATISFIED

- METRICS AVAILABLE TO JUDGE QUALITY OF STRUCTURAL DECOMPOSITION
  - COUPLING - RELATIONSHIPS AMONG MODULES
  - COHESION - RELATEDNESS WITHIN A MODULE

INSTRUCTOR NOTES

AS THE NAME OF THIS TECHNIQUE IMPLIES STRUCTURING ACCORDING TO WELL DEFINED RULES IS ONE OF THE IMPORTANT UNDERLYING PRINCIPLES - THE MODULAR STRUCTURE USES ABSTRACTION AND GRAPHICAL REPRESENTATIONS TO AID IN THE UNDERSTANDABILITY OF THE SYSTEM.

THE DATA FLOW GRAPHS AND STRUCTURE CHARTS HAVE WELL DEVELOPED SYNTAX AND SEMANTICS RULES REFLECTING A REASONABLE (BUT NOT EXTREME) DEGREE OF UNIFORMITY AND FORMALISM.

THE COUPLING AND COHESION METRICS ARE SPECIFICALLY AIMED AT INSURING THAT THE DESIGN AVOIDS PROBLEMS THAT WOULD IMPACT ITS MODIFIABILITY.

STRUCTURED DESIGN

PRINCIPLES

- STRUCTURING, MODULARITY
  - PROCESS DEVELOPS MODULAR, HIERARCHICAL STRUCTURE
- ABSTRACTION
  - DATA FLOW GRAPH'S DECOMPOSITION AND STRUCTURE CHART'S HIERARCHY BOTH SUPPORT ABSTRACTION
- UNIFORMITY, FORMALISM
  - BOTH DATA FLOW GRAPHS AND STRUCTURE CHARTS HAVE WELL-DEVELOPED SYNTAX AND SEMANTIC RULES

GOALS

- UNDERSTANDABILITY
  - GRAPHIC NATURE OF TECHNIQUES AIDS UNDERSTANDING
- MAINTAINABILITY, MODIFIABILITY
  - STRATEGIES ARE INTENDED TO PRODUCE MAINTAINABLE DESIGNS
  - COUPLING AND COHESION RULES SUPPORT MODIFIABILITY
- TRACEABILITY
  - STRATEGIES PROVIDE TRACEABLE DESIGN

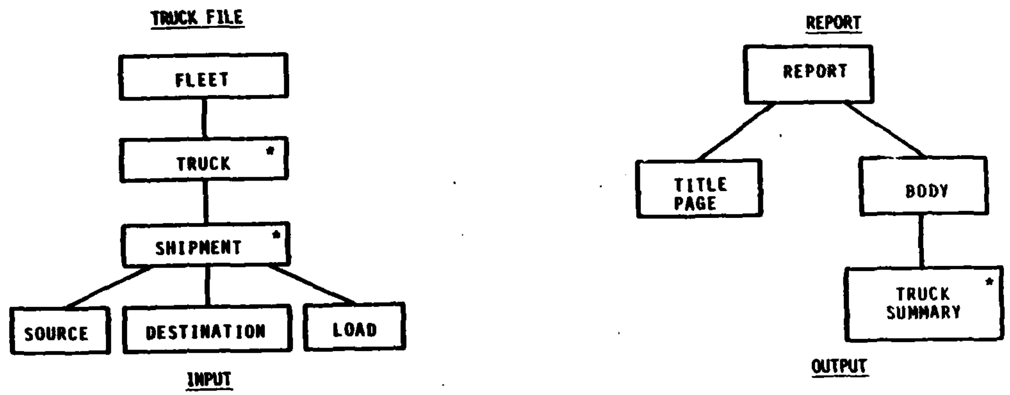
INSTRUCTOR NOTES

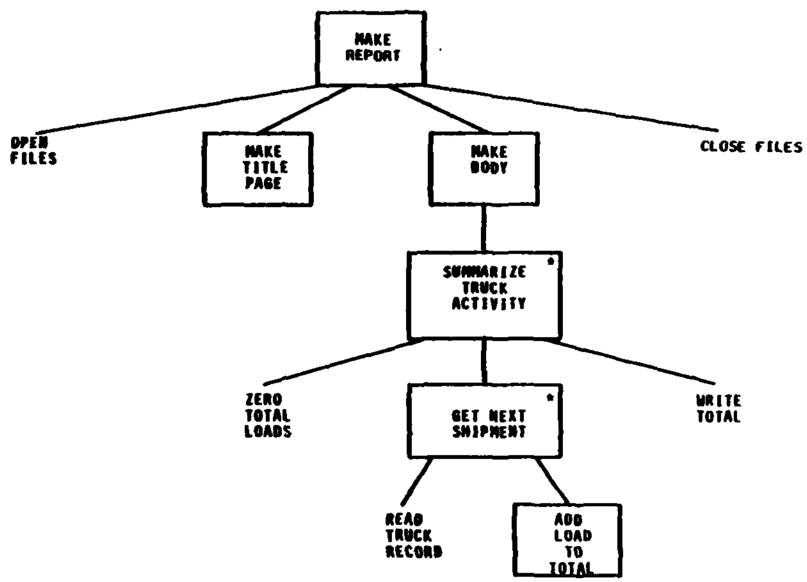
THE JACKSON AND WARNIER-ORR TECHNIQUES ARE VERY SIMILAR IN THEIR UNDERLYING PHILOSOPHY, BUT USE SIGNIFICANTLY DIFFERENT REPRESENTATIONS.

BOTH TECHNIQUES ANALYZE THE STRUCTURE OF THE INPUT AND OUTPUT DATA AND THEN PROCEED TO PRODUCE A DESIGN WHICH MAPS THE INPUT STRUCTURE INTO THE OUTPUT STRUCTURE.

POINT OUT THE SPECIAL SYNTAX - THE * MEANS REPETITION, THE N AND M IMPLY VARIABLE AMOUNTS ARE TO BE HANDLED.

JACKSON/WARNIER-ORR





JACKSON

INSTRUCTOR NOTES

JACKSON/WARNIER-ORR (Continued)

- **INPUT:**

FILE { PART (N) { MOVEMENT RECORD (M) { ISSUE ⊕ RECEIPT

- **OUTPUT:**

REPORT { HEADING / BODY { NET MOVEMENT LINE (X)

MAKE REPORT {
- OPEN FILES
- MAKE HEADING
- MAKE BODY { MAKE NET MOVEMENT LINE (N) {
  - ZERO NET MOVEMENT
  - PROCESS MOVEMENT RECORD (M) {
    - READ NEXT RECORD
    - UPDATE NEXT MOVEMENT {
      - SUBTRACT FROM NET MOVEMENT
      - ⊕
      - ADD TO NET MOVEMENT
  - WRITE NET LINE
- CLOSE FILES

WARNIER-ORR

INSTRUCTOR NOTES

BOTH OF THESE TECHNIQUES STRESS THE PHILOSOPHY THAT THE ULTIMATE PROGRAM STRUCTURE SHOULD MATCH THE DATA STRUCTURE AS MUCH AS POSSIBLE.

THEREFORE YOU START BY ANALYZING (DESCRIBING) THE INPUT AND OUTPUT DATA STRUCTURES. SOMETIMES A PROGRAM STRUCTURE WHICH MAPS FROM ONE TO THE OTHER IS STRAIGHT FORWARD - JUST LIKE CONNECTING THE OUTPUT STRUCTURE TO THE INPUT STRUCTURE. WHEN THIS ISN'T THE CASE WE HAVE A STRUCTURE CLASH AND USE AN INTERMEDIATE FILE, PROGRAM INVERSION (ALMOST TURNING THE INPUT OR OUTPUT STRUCTURE UPSIDE DOWN) OR EVEN TASKING AND INTERMEDIATE BUFFERS.

WARNIER-ORR WAS DEVELOPED INDEPENDENTLY IN THE SAME TIME PERIOD AND USES A MORE TEXTUAL APPROACH.

BOTH TECHNIQUES HAVE BEEN APPLIED PRIMARILY TO COMMERCIAL (COBOL) APPLICATIONS, BUT HAVE BEEN EXTENDED TO REAL TIME SYSTEMS.

JACKSON/WARNIER-ORR

- JACKSON DESIGN DEVELOPED BY MICHAEL JACKSON AND IS WIDELY USED IN EUROPE
- CONSISTS OF TWO ASPECTS:
  - JACKSON STRUCTURED PROGRAMMING (JSP) (DESCRIBED HERE)
  - JACKSON SYSTEM DEVELOPED (JSD) (A MORE RECENT ADDITION FOR HIGHER LEVEL SYSTEM DESIGN ISSUES)
- KEY CONCEPT
  - PROGRAM STRUCTURE SHOULD MIRROR DATA STRUCTURE WHICH MIRRORS PROBLEM STRUCTURE
- WHEN INPUT STRUCTURE AND OUTPUT STRUCTURE DON'T MATCH - RESOLVED WITH STRUCTURE CLASH USING INTERMEDIATE FILES AND PROGRAM INVERSION
- WARNIER-ORR DEVELOPED IN FRANCE BY JEAN WARNIER
  - POPULARIZED IN THIS COUNTRY BY KEN ORR
- WARNIER-ORR USES MUCH THE SAME PHILOSOPHY AS JSP WITH TEXTUAL REPRESENTATION

INSTRUCTOR NOTES

TECHNIQUES PROVIDE A COOKBOOK APPROACH FOR DEVELOPING A MODULAR STRUCTURE BASED ON DATA STRUCTURES. HIGH DEGREE OF TRACEABILITY, AND UNDERSTANDABILITY IS INHERENT IN THE STRAIGHT FORWARD COOKBOOK APPROACH.

CAN BE A PRODUCTIVE TECHNIQUE FOR DATA PROCESSING APPLICATIONS.

WARNIER-ORR

PRINCIPLES

- STRUCTURING, MODULARITY
  - DERIVED FROM DATA STRUCTURE
- UNIFORMITY, FORMALISM
  - SYNTAX AND SEMANTIC RULES FOR HANDLING COMPLEX DATA STRUCTURES

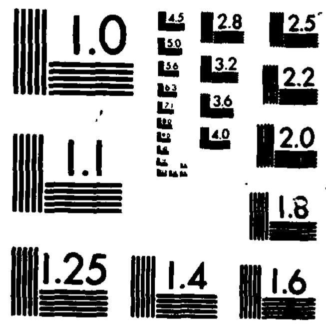
GOALS

- UNDERSTANDABILITY, TRACEABLE
  - STRUCTURE OF PROGRAM RELATES DIRECTLY TO STRUCTURE OF DATA
- PRODUCTIVITY
  - A COOKBOOK APPROACH FOR DATA HANDLING APPLICATIONS
- MAINTAINABILITY, MODIFIABILITY
  - CHANGES TO INPUT OR OUTPUT MAP DIRECTLY TO CORRESPONDING PROGRAM STRUCTURE CHANGES

AD-A165 123 ADA (TRADEMARK) TRAINING CURRICULUM: SOFTWARE ENGINEERING FOR MANAGERS M101 TEACHER'S GUIDE(U) SOFTECH INC WALTHAM MA 1986 DAAB07-83-C-K506

3/4

UNCLASSIFIED F/G 5/9 NL



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

INSTRUCTOR NOTES

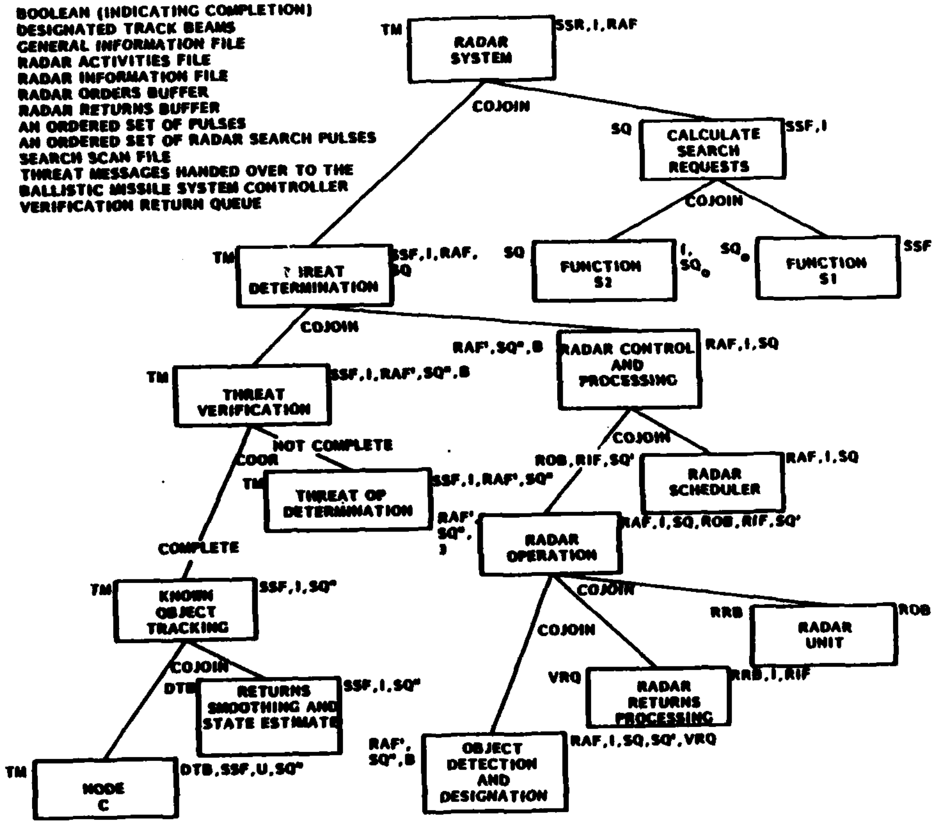
HOS BUILDS PROGRAM STRUCTURES OUT OF A LIMITED NUMBER OF PROVABLE CORRECT PRIMITIVES. IT USES A LIMITED NUMBER OF PRIMITIVE CONSTRUCTS (JOIN, OR, INCLUDE) AND NON-PRIMITIVE (CONCUR, COJOIN, COOR (NOT A BEER)). EACH OF THESE HAS VERY SPECIFIC RULES CONCERNING HOW THE INPUTS AND OUTPUTS OF THE LOWER LEVEL ARE PRODUCED OR USED AT THE PARENT LEVEL.

HOS

RADAR SYSTEM

EXAMPLE

- B: BOOLEAN (INDICATING COMPLETION)
- DTB: DESIGNATED TRACK BEAMS
- I: GENERAL INFORMATION FILE
- RAF: RADAR ACTIVITIES FILE
- RIF: RADAR INFORMATION FILE
- ROB: RADAR ORDERS BUFFER
- RRB: RADAR RETURNS BUFFER
- SQ_e: AN ORDERED SET OF PULSES
- SQ: AN ORDERED SET OF RADAR SEARCH PULSES
- SSF: SEARCH SCAN FILE
- TM: THREAT MESSAGES HANDED OVER TO THE BALLISTIC MISSILE SYSTEM CONTROLLER
- VRQ: VERIFICATION RETURN QUEUE



SOURCE: MARTIN, SYSTEM DESIGN FROM PROVABLY CORRECT CONSTRUCTS, 1985

INSTRUCTOR NOTES

- HOS HELPS MODEL SYNCHRONOUS, ASYNCHRONOUS, NETWORKED, REAL TIME, INTERRUPT-DRIVEN, RECURSIVE, AND INTERACTIVE SYSTEMS.

- IT HAS BEEN AROUND IN SOME FORM SINCE THE MID-60's, AND HAS BEEN USED BY NASA SINCE THE APOLLO SPACE PROGRAM. EXTENSIVE USE IN SHUTTLE PROGRAM.

- IT IS VERY HIERARCHAL IN NATURE.

- AXIOMS CONTROL DECOMPOSITIONS. PARENTS CAN INVOKE ONLY ITS OFFSPRING, CONTROLS THE ORDER OF THEIR INVOCATION, AND CONTROLS THEIR INPUT AND OUTPUT (I.E., ACCESS RIGHTS).

- ALL CONTROL STRUCTURES ARE DERIVED FROM THE AXIOMS.

- DATA TYPES AUTOMATICALLY DEFINE PRIMITIVE OPERATIONS ON DATA.

- FUNCTIONS REPRESENT SPECIFIC TRANSFORMATIONS OF INPUT VALUES OF DATA TYPE MEMBERS TO OUTPUT VALUES OF DATA TYPE MEMBERS.

- THE AXIOMATIC NATURE OF HOS ALLOWS THE INPUTS TO BE TRANSFORMED INTO REPRESENTATIONS TO BE USED AS INPUTS TO JACKSON, PSL/PSA, SADT, SREM, AND PETRI NETS.

HOS

- GREW FROM THE APOLLO MOON MISSION
- DEVELOPED FOR REAL TIME SYSTEMS
- EMPHASIS IS ON GENERATION OR PROVABLY CORRECT SOFTWARE
- BASED ON A SET OF SIX AXIOMS
- SOFTWARE SYSTEM CAN BE REPRESENTED AS A HIERARCHICAL SYSTEM MODEL BASED ON THESE AXIOMS
- HIERARCHY USED WITH PROGRAM GENERATOR TO "AUTOMATICALLY" PRODUCE "CORRECT" PROGRAMS

INSTRUCTOR NOTES

SOFTWARE FOR APPLYING SUCH TECHNIQUES REQUIRES THE FOLLOWING COMPONENTS

1. A LANGUAGE FOR EXPRESSING FUNCTIONS AND THEIR DECOMPOSITION INTO OTHER FUNCTIONS.

2. AN INTERACTIVE SCREEN FACILITY FOR CONSTRUCTING AND MANIPULATING THE CONTROL MAPS, AND ALLOWING THE USER TO CORRECT ERRORS INTERACTIVELY.

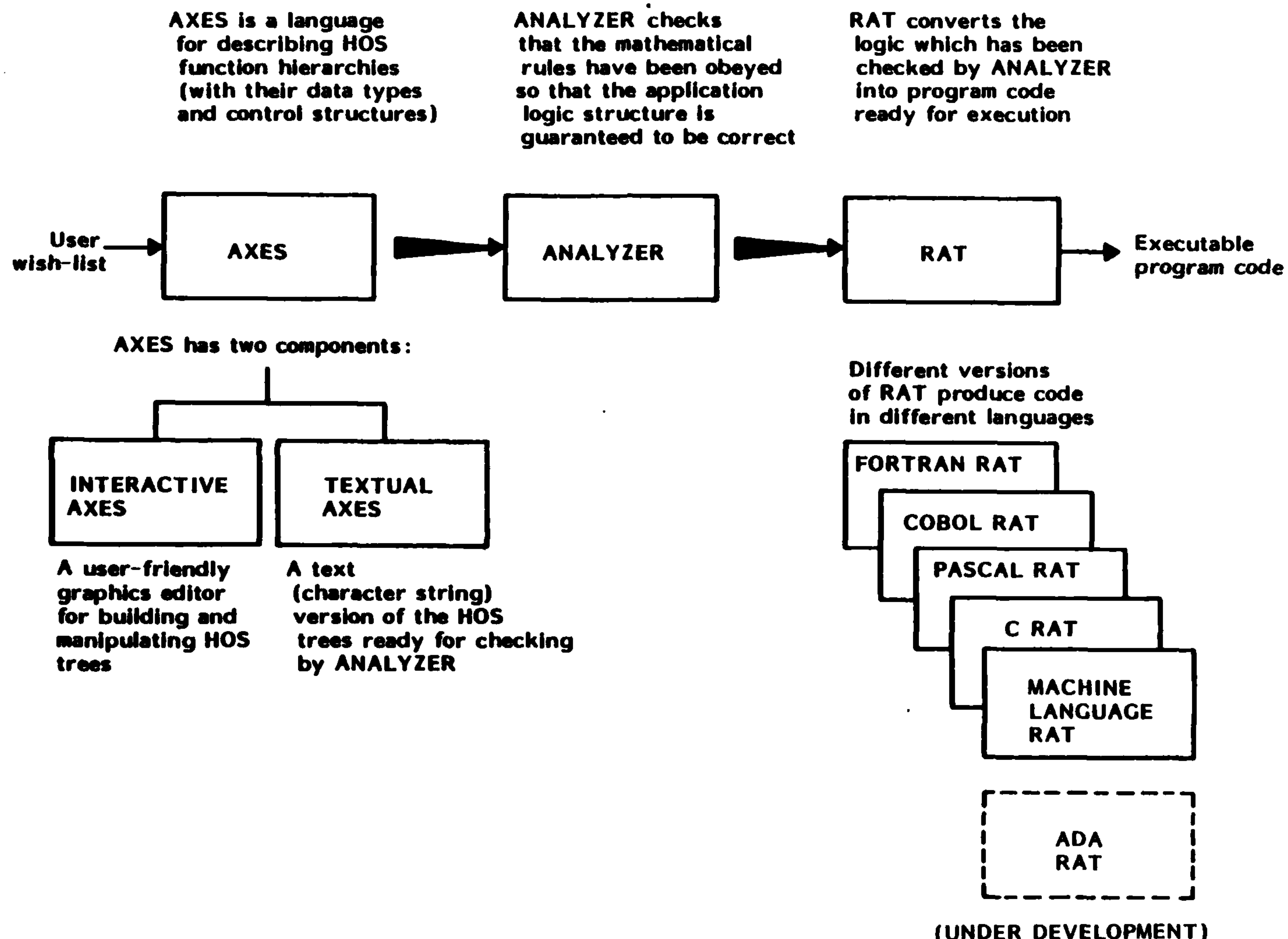
3. A LIBRARY OF DATA TYPES, PRIMITIVE FUNCTIONS, AND PREVIOUSLY DEFINED MODULES.

4. AN ANALYZER ROUTINE FOR AUTOMATICALLY CHECKING THAT ALL THE RULES THAT GIVE PROVABLY CORRECT LOGIC HAVE BEEN FOLLOWED.

5. A GENERATOR THAT AUTOMATICALLY GENERATES PROGRAM CODE.

AUTOMATION OF HOS

(USE.IT)



SOURCE: MARTIN, SYSTEM DESIGN FROM PROVABLY CORRECT CONSTRUCTS, 1985

INSTRUCTOR NOTES

HOS IS THE MOST FORMAL OF ANY OF THE TECHNIQUES WE HAVE LOOKED AT AND IS ABLE TO PROVIDE THE MOST ASSISTANCE IN PRODUCING CORRECT, VERIFIABLE, AND RELIABLE SYSTEMS. PRODUCTIVITY IS ENHANCED BY THE AUTOMATED USE.IT TOOL.

HOS

PRINCIPLES

- FORMALISM, UNIFORMITY
  - PROVABLY CORRECT AXIOMATIC BASIS
- STRUCTURING, MODULARITY
  - HIERARCHICAL STRUCTURE BASED ON STRICT DECOMPOSITION RULES

GOALS

- CORRECTNESS, VERIFIABILITY
  - FORMAL, MATHEMATICAL BASIS FACILITATES VERIFYING CORRECTNESS
- RELIABILITY
  - ORIENTED SPECIFICALLY FOR HIGH RELIABILITY SYSTEMS
- PRODUCTIVITY
  - AUTOMATION ASSISTS IN GENERATING CODE FROM DESIGN DESCRIPTION

INSTRUCTOR NOTES

SECTION 11

DESIGN METHOD OVERVIEW

INSTRUCTOR NOTES

WE'RE PRESENTING THREE DETAILED DESIGN TECHNIQUES. TWO, HIPO AND NSSF HAVE BEEN AROUND A WHILE AND ARE BASICALLY GRAPHICAL IN NATURE.

PDL HAS EVOLVED FROM OLD PSUEDO-CODE TECHNIQUES AND IS BECOMING A HIGHLY VISIBLE APPROACH THAT MAY WELL BE MANDATED ON MANY PROJECTS.

DETAILED DESIGN METHOD OVERVIEW

- PDL - PROGRAM DESIGN LANGUAGE
- HIPO - HIERARCHAL INPUT PROCESSING OUTPUT
- NSSF - NASSI-SCHNEIDERMAN STRUCTURED FLOWCHARTS

INSTRUCTOR NOTES

WE AREN'T GOING TO DEFINE A SINGLE PDL. WE ARE GOING TO GIVE YOU AN INTRODUCTION TO THE ISSUES INVOLVED WITH DEFINING AND USING PDL'S.

THE DIFFICULTY IN UPDATING FLOWCHARTS REALLY LED TO PDL. IN MAINTAINING A SYSTEM IT IS FOOLISH TO RELY ON FLOWCHARTS. SO A MECHANISM WAS NEEDED TO: 1) DESCRIBE THE HIGH LEVEL STRUCTURE AND 2) BE UPDATED EASILY.

STATE THAT MANY ELEMENTS OF DESIGN ARE GRAPHIC IN NATURE. TEXTUAL LANGUAGES, LIKE PDL, NATURALLY FALL SHORT OF MEETING ALL THE NEEDS OF A DESIGNER.

PROGRAM DESIGN LANGUAGE OVERVIEW

- IS A TEXTUAL LANGUAGE, PRECISE ENOUGH TO DESCRIBE SOFTWARE, YET EXPRESSIVE ENOUGH TO DESIGN WITH.

PRECISION ⟷ EXPRESSION

ALWAYS COMPETE.

PROPER BALANCE IS UNKNOWN.

- PDL'S GREW OUT OF THE PROBLEMS ON TRYING TO KEEP UP-TO-DATE FLOWCHARTS OF THE SYSTEM.

- PDL'S WILL BE REQUIRED FOR DOCUMENTING DETAIL DESIGNS.

INSTRUCTOR NOTES

A DESIGN LANGUAGE IS A <u>TEXTUAL</u> REPRESENTATION FOR THE PRECISE EXPRESSION OF PROGRAM OR SYSTEM DESIGNS.

IN USING THE IEEE PDL STANDARD, THERE IS A "RECOMMENDED PRACTICE" THAT GOES ALONG WITH IT. IT PROVIDES GENERAL GUIDANCE IN USING THE PDL, THE FEATURES OF AN Ada PDL, DESIGN LANGUAGE CHARACTERISTICS, MANAGEMENT ISSUES, ETC.

Ada AS A PROGRAM DESIGN LANGUAGE

- IEEE HAS PRODUCED A DRAFT SET OF GUIDELINES FOR USE OF Ada AS A DESIGN LANGUAGE.
  - ONLY "REAL" STANDARD FOR MODERN PDL.
  - USED AS A BASIS FOR THIS SECTION

- IEEE MOTIVATING GOALS FOR USE OF Ada AS A PDL ARE TO:
  - UTILIZE THE POWER OF THE Ada PROGRAMMING LANGUAGE IN THE DESIGN PROCESS.
  - ENHANCE COMMUNICATION BY USING THE SAME LANGUAGE NOTATION THROUGHOUT THE LIFE CYCLE.
  - SUPPORT QUALITY SOFTWARE DESIGN BY FOCUSING ON APPROPRIATE LEVELS OF DESIGN DETAIL.
  - CAPITALIZE ON THE EMERGING AVAILABILITY OF Ada TOOLS AND INDUSTRY SUPPORT FOR THE Ada LANGUAGE.
  - PROVIDE A MECHANISM THAT SUPPORTS THE TRANSITION TO Ada BASED SOFTWARE ENGINEERING PRACTICE.
  - PROVIDE A BASIS FOR STANDARDIZATION.

THE USE OF A DESIGN LANGUAGE CAN INCREASE PRODUCTIVITY IN A NUMBER OF WAYS. FIRST, AS LONG AS DESIGN DOCUMENTATION CAN BE WRITTEN IN MACHINE READABLE FORM, AUTOMATED TOOLS CAN BE CREATED TO CHECK FOR COMPLETENESS AND CONSISTENCY, AND TO AUTOMATE DEVELOPMENT ACTIVITIES. SECOND, SUCH A LANGUAGE FACILITATES COMMUNICATION BETWEEN VARIOUS MEMBERS OF A PROJECT TEAM. FINALLY, IT IS ADVANTAGEOUS TO HAVE A SINGLE NOTATION THAT CAN BE USED THROUGHOUT ALL ACTIVITIES OF THE SOFTWARE LIFE CYCLE.

USE OF A DESIGN LANGUAGE CAN INCREASE SOFTWARE QUALITY BY FACILITATING THE EARLY DETECTION AND CORRECTION OF ERRORS BY HELPING TEAM MEMBERS TO COMMUNICATE THROUGH A COMMON LANGUAGE, AND BY PROVIDING A MECHANISM FOR VERIFICATION OF DESIGN AND THE TRANSLATION OF THE DESIGN NOTATION. A DESIGN LANGUAGE PROVIDES A VEHICLE BOTH FOR COMMUNICATIONS AND FOR SUPPORTING VARIOUS ACTIVITIES OF THE PRODUCT LIFE CYCLE, AND SHOULD BE HUMAN ENGINEERED, PRECISE, ANALYZABLE, VERIFIABLE AND SUPPORTIVE OF VARIOUS PROGRAMMING METHODOLOGIES.

RATIONALE FOR USING A PDL

- TO INCREASE PRODUCTIVITY
  - PDLs ARE MACHINE READABLE; TOOLS CAN CHECK FOR COMPLETENESS, CONSISTENCY AND CONFORMANCE TO STANDARDS.
  - PDLs DEFINE A COMMON LANGUAGE TO ENHANCE COMMUNICATIONS WITHIN A PROJECT.

- TO IMPROVE SOFTWARE QUALITY
  - PDLs FACILITATE EARLY DETECTION AND CORRECTION OF ERRORS BY AIDING IN COMMUNICATION AND VERIFICATION.

- TO MINIMIZE THE RISKS INVOLVED IN DEVELOPING AND MAINTAINING SOFTWARE
  - PDLs MAKE DESIGNS VISIBLE EARLY.
  - COMMON PDL AND IMPLEMENTATION LANGUAGE REDUCES TRAINING NEEDS.

INSTRUCTOR NOTES

A DESIGN METHODOLOGY IS A BODY OF PROCEDURES AND TECHNIQUES WHICH CAN BE USED TO CONSTRUCT A SYSTEM DESIGN. THE DESIGN ITSELF IS AN ABSTRACTION OF THE PROPOSED SYSTEM; WHEN THE SYSTEM IS IMPLEMENTED ACCORDING TO THE DESIGN, IT IS EXPECTED TO PERFORM ACCORDING TO THE SYSTEM SPECIFICATION AND TO SATISFY THE SYSTEM REQUIREMENTS. THE DESIGN LANGUAGE SHOULD SUPPORT THE DESCRIPTION OF A PRODUCT AT THE APPROPRIATE LEVELS OF DETAIL FOR EACH ACTIVITY OF A GIVEN METHODOLOGY.

PDL REQUIREMENTS

- A PDL MUST SUPPORT ...
  - ABSTRACTION - EMPHASIZING PARTICULAR CONCEPTS WHILE SUPPRESSING UNNECESSARY DETAIL.
  - DECOMPOSITION - DIVIDING A LARGE SYSTEM INTO SMALLER, MORE MANAGEABLE PIECES WHILE MAINTAINING A FIXED LEVEL OF DETAIL.
  - INFORMATION HIDING - ISOLATING DESIGNATED INFORMATION TO CONTROL THE DEPENDENCE ON A PARTICULAR IMPLEMENTATION.
  - STEPWISE REFINEMENT - PROGRESSIVELY ADDING DETAIL TO A DESIGN DESCRIPTION.
  - MODULARIZATION - ISOLATING PORTIONS OF A SYSTEM INTO LOGICALLY SEPARABLE PARTS SUCH THAT A CHANGE IN ONE PART HAS MINIMAL IMPACT ON OTHER PARTS.

NOTE: THESE ARE SOME OF THE DOD'S REQUIREMENTS FOR ADA

INSTRUCTOR NOTES

THESE CHARACTERISTICS ARE DEPENDENT UPON THE METHODOLOGY USED.

PDL CHARACTERISTICS

- A PROGRAM DESIGN LANGUAGE SHOULD ADDRESS ALL OF THE FOLLOWING:
  - EXPRESSIVE POWER
    - ALGORITHM DESIGN
    - DATA STRUCTURES
    - SUPPLEMENTARY INFORMATION
    - CONNECTIVITY
    - MANAGEMENT OF COMPLEXITY
  - HUMAN FACTORS
  - ANALYZABILITY
  - RELATIONSHIP BETWEEN THE PDL AND IMPLEMENTATION LANGUAGE
  - TOOL IMPACT ON THE PROGRAM DESIGN LANGUAGE
  - IMPLEMENTATION CONSIDERATIONS

INSTRUCTOR NOTES

USE THIS SLIDE TO SHOW THE READABILITY OF AN Ada PDL AND THE TRACEABILITY TO THE Ada IMPLEMENTATION.

NOTATION

// ~~~~~~~~~~~~~~~~ // AN ILLEGAL Ada CONSTRUCT TO EXPRESS DESIGN INFORMATION THAT WILL BE LATER EXPANDED OR THAT IS MORE INFORMATIVE EXPRESSED IN ENGLISH THAN IN Ada.

AN ADA PDL (SAMPLE)

DESIGN

```
procedure SORT (TABLE : in out TABLE_TYPE) is
begin
   if // TABLE has more than one entry // then
         while // TABLE is not sorted // loop
            for // each pair of entries in TABLE // loop
                  if // 1st entry is greater than 2nd
                        then
                     // exchange the two entries//;
                  end if;
            end loop;
         end loop;
   end if;
end SORT;
```

EXPANDS TO

IMPLEMENTATION

```
procedure SORT (TABLE : in out TABLE_TYPE) is
   SWAPPED_ITEMS : Boolean;
   TEMP : ITEM_TYPE;
begin
   if TABLE'LENGTH > 1 then
      SWAPPED_ITEMS := TRUE;
      -- loop while the table is not sorted
      while not SWAPPED_ITEMS loop
         SWAPPED_ITEMS := FALSE;
         for I in TABLE'FIRST..INDEX_TYPE'PRED(TABLE'LAST)
         loop
            if TABLE (I) > TABLE(INDEX_TYPE'SUCC(I)) then
               SWAPPED_ITEMS := TRUE;
               TEMP := TABLE(I);
               TABLE(I) := TABLE(INDEX_TYPE'SUCCC(I))
               TABLE(INDEX_TYPE'SUCC(I)) := TEMP;
            end if;
         end loop; -- for each pair of entries
      end loop; -- for sorting TABLE
   end if;
end SORT;
```

INSTRUCTOR NOTES

- IF NO ONE CHIMES UP FOR DISCUSSION, ASK THE QUESTION "SHOULD Ada BE USED AS A PDL?"

PDL SUMMARY

- PDL'S HAVE A PLACE IN DESIGN, THEY ...
  - CAN INCREASE PRODUCTIVITY
  - CAN IMPROVE QUALITY
  - CAN MINIMIZE RISK
  - WILL BE REQUIRED BY DOD-STD-SDS.

- PDL'S HAVE DRAWBACKS ...
  - PEOPLE WILL CODE INSTEAD OF DESIGN
  - FULLY COMPILABLE PDLS ARE NOT EXPRESSIVE ENOUGH FOR REAL TIME DESIGN

PDL IS REALLY A TOOL NOT A TECHNIQUE - IT IS RELATIVELY FORMAL AND UNIFORM, ESPECIALLY IF COMPATIBILITY WITH Ada SYNTAX RULES IS A REQUIREMENT. THE ABILITY TO ATTAIN MODULARITY, STRUCTURING, AND ABSTRACTION ARE INHERITED FROM THE LANGUAGE BASE FOR THE PDL. THESE AND OTHER PRINCIPLES, SUCH AS HIDING AND SEPARATION OF CONCERNS, CAN BE UTILIZED BUT ARE NOT REALLY FACILITATED OR ENFORCED THROUGH ANY METHODOLOGY. VARIOUS METHODOLOGIES, SUCH AS OBJECT ORIENTED DESIGN, WILL PROBABLY BE CREATED TO ADD A RECOMMENDED APPROACH TO USING THIS TOOL.

BECAUSE PDL IS ONLY A TOOL, IT ALONE DOES NOT HELP US IN ATTAINING MANY OF OUR GOALS. ONLY THOSE THAT DERIVE FROM ITS FORMALITY, SUCH AS VERIFIABILITY AND CORRECTNESS, ARE EASILY ATTAINED. UNDERSTANDABILITY IS IMPROVED OVER THE HIGH LEVEL LANGUAGE ITSELF, BUT STILL SUFFERS WHEN COMPARED TO GRAPHICAL OR MORE ABSTRACT TECHNIQUES.

PDL

PRINCIPLES

- FORMALISM, UNIFORMITY
  - ALMOST AS HIGH AS FOR A PROGRAMMING LANGUAGE
- MODULARITY, STRUCTURING, ABSTRACTION
  - SAME AS FOR A HIGH LEVEL LANGUAGE
- HIDING, SEPARATION OF CONCERNS
  - *CAN BE SUPPORTED, DON'T COME* AUTOMATICALLY

GOALS

- VERIFIABILITY, CORRECTNESS
  - CAN USUALLY BE MACHINE PROCESSED
- UNDERSTANDABLE
  - IMPROVED OVER HIGH-LEVEL LANGUAGE

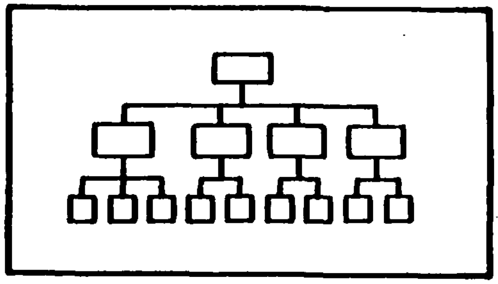
INSTRUCTOR NOTES

THE EMPHASIS WITH HIPO IS ON GRAPHICS TO AID UNDERSTANDABILITY. THE HIERARCHY IS REPRESENTED BY A HIERARCHAL VISUAL TABLE OF CONTENTS (VTOC). THESE ARE SIMILAR TO, BUT NOT AS REFINED AS THE STRUCTURED ANALYSIS STRUCTURE CHARTS.

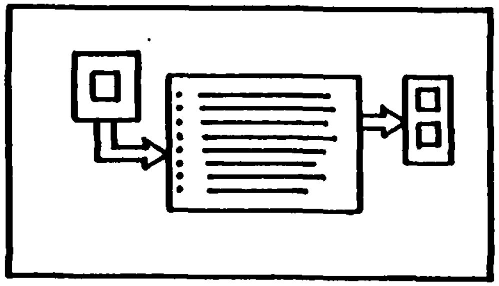
THE OVERALL PROCESSING OF THE SYSTEM IS DESCRIBED IN AN OVERVIEW DIAGRAM WHICH SHOWS INPUTS, PROCESSING, AND OUTPUTS. THESE OVERVIEW DIAGRAMS SOMETIMES USE RELATIVELY SOPHISTICATED GRAPHICS. EACH MODULE (FROM THE VTOC) IS REPRESENTED BY A DETAILED DIAGRAM. THESE HAVE LESS GRAPHICS BUT STILL USE BOLD ARROWS TO ASSOCIATE INPUTS AND OUTPUTS WITH PROCESSING.

HIPO

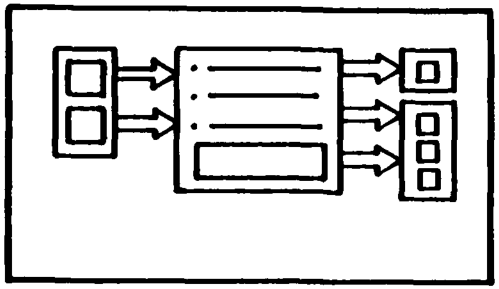
1 A VISUAL TABLE OF CONTENTS
(HIERARCHY)



2 OVERVIEW DIAGRAMS
(FOR HIGH LEVELS IN THE VTOC)



3 DETAIL DIAGRAMS
(FOR LOW LEVELS IN THE VTOC)



INSTRUCTOR NOTES

HIPO WAS DEVELOPED IN THE EARLY 70's BY IBM. IT WAS USED MORE TO DOCUMENT STABLE SOFTWARE FOR UNDERSTANDABILITY BY MAINTAINERS OR USERS THAN AS A TECHNIQUE USED BY DESIGNERS IN THE MIDDLE OF A DEVELOPMENT CYCLE. THIS IS BECAUSE THE GRAPHICS ARE RELATIVELY ELABORATE AND COSTLY TO PRODUCE.

THE EXACT TECHNIQUE HASN'T REALLY CAUGHT ON OUTSIDE OF IBM BUT MANY VARIATIONS HAVE BEEN DEVELOPED, STILL CALLED HIPO, BUT WITH LESS GRAPHICS AND POSSIBLY THE USE OF A PDL FOR THE PROCESSING DESCRIPTION.

HIPO

- INTRODUCED BY IBM
- USED MORE AS POST DEVELOPMENT DOCUMENTATION THAN AS A PREDEVELOPMENT DESIGN AID
- EMPHASIS IS ON UNDERSTANDABILITY
- DIAGRAMS ARE RELATIVELY COSTLY TO PRODUCE
- TECHNIQUE HAS BEEN ADAPTED BY OTHERS WITH LESS EMPHASIS ON GRAPHICS AND PDL FOR PROCESSING DESCRIPTION

INSTRUCTOR NOTES

THE PRIMARY USE OF THE ORIGINAL IBM TECHNIQUE WAS AIMED AT IMPROVING THE MAINTAINABILITY OF COMPLEX SYSTEMS BY IMPROVING THE UNDERSTANDABILITY OF THEIR DOCUMENTATION. THIS TECHNIQUE DOES USE HIERARCHICAL STRUCTURING AND ENCOURAGES MODULARITY BUT DOESN'T PROVIDE ANY ADDITIONAL MODULARIZATION CRITERIA - IT CAN AND HAS BEEN USED WITH OTHER DESIGN METRICS (COHESION AND BINDING) AS THE MODULARIZATION CRITERIA. ONE OF THE BIGGEST AIDS TO UNDERSTANDABILITY IS THE USE OF GRAPHICS - THIS ALSO MAKES IT COSTLY AND IS PROBABLY WHY IT HAS NOT BECOME WIDESPREAD IN ITS ORIGINAL FORM.

HIPO

PRINCIPLES

- STRUCTURING MODULARITY
  - VTOC PROVIDES MECHANISM
- ABSTRACTION
  - GRAPHICS AND VTOC SUPPORT THIS

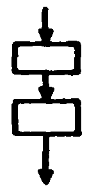
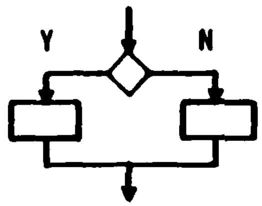
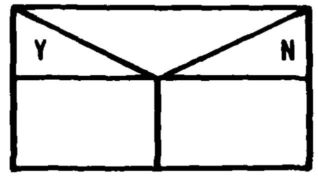
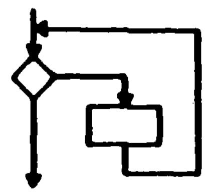
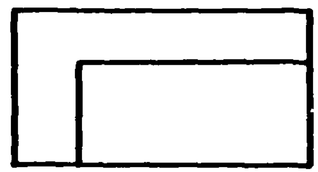
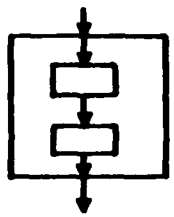
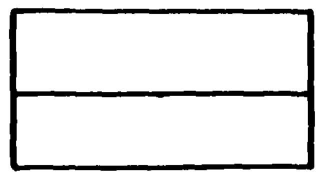
GOALS

- UNDERSTANDABILITY, MAINTAINABILITY
  - MAJOR EMPHASIS IS ON PRODUCING UNDERSTANDABLE DOCUMENTATION FOR MAINTENANCE
- TRACEABLE
  - MULTIPLE LEVELS OF DESCRIPTION PROVIDE SOME DEGREE OF ASSISTANCE

INSTRUCTOR NOTES

- COMPARE NSSF TO CONVENTIONAL FLOWCHART GRAPHICS.

- BE CAREFUL OF THE HIERARCHY EXAMPLE: THINK OF IT AS LOOKING "INSIDE" THE BIG BOX.

STRUCTURE	FLOWCHART GRAPHICS	NSSF GRAPHICS
SEQUENCE		
SELECTION		
REPETITION		
HIERARCHY		

INSTRUCTOR NOTES

NSSF IS A WAY OF DESCRIBING THE ALGORITHMIC STRUCTURE OF A PROGRAM. IT WORKS WELL ONLY ON STRUCTURED PROGRAMMING.

NSSF

- NASSI-SCHNEIDERMAN STRUCTURED FLOWCHARTS
  - DOCUMENT THE <u>PROCEDURAL DETAILS</u> OF SOFTWARE WITH IMPROVED FLOWCHART GRAPHICS
- BASED ON THE ASSUMPTION THAT FLOW CHARTING GRAPHICS CAN BE <u>SIMPLIFIED</u> TO REPRESENT <u>STRUCTURED PROGRAMMING CONSTRUCTS</u>
- NSSF GRAPHICS HELP DESIGNERS
  - <u>REPRESENT PROGRAM STRUCTURES</u>
  - <u>IDENTIFY COMPLEXITY</u> IN A MODULE

INSTRUCTOR NOTES

NSSF IS VERY USEFUL IN TEACHING STRUCTURED PROGRAMMING AND FOR DOCUMENTATION. IT IS EXPENSIVE TO PRODUCE, THUS AN AUTOMATED TOOL IS NEEDED TO EXTRACT THE NSSF FROM THE CODE. WITHOUT SUCH A TOOL, IT WILL BE DIFFICULT TO KEEP THEM UP TO DATE.

AGAIN THIS IS ONLY A TOOL NOT A TECHNIQUE OR METHODOLOGY. NSSF IS PROBABLY ONE OF THE MOST UNDERSTANDABLE TOOLS FOR REPRESENTING FLOW OF CONTROL (AT A DETAILED LEVEL) AVAILABLE.

NSSF

PRINCIPLES

- STRUCTURING
  - AT A LOW LEVEL, PROGRAM STRUCTURING ONLY
- UNIFORMITY, FORMALISM
  - OF THE GRAPHICAL CONSTRUCTS

GOALS

- UNDERSTANDABILITY
  - REPRESENTS PROGRAM FLOW OF CONTROL GRAPHICALLY

INSTRUCTOR NOTES

THEME: IMPLEMENTATION IS MORE THAN JUST CODING!

PURPOSE: TO IDENTIFY THE KEY ISSUES ONE MUST CONSIDER DURING THE IMPLEMENTATION PHASE OF THE LIFE CYCLE.

REFERENCE: DOD-STD-SDS

SECTION 12

IMPLEMENTATION OVERVIEW

INSTRUCTOR NOTES

THE IMPLEMENTATION PHASE SHOULD CREATE A SYSTEM WHICH IMPLEMENTS THE DESIGN CORRECTLY AND MEETS HE RESOURCE, ACCURACY, AND PERFORMANCE CONSTRAINTS IN THE SPECIFICATION. IMPLEMENTATION TRANSLATES DESIGN INTO THE LANGUAGE OF THE TARGET COMPUTER, THE DATABASE, AND OTHER PRODUCTS NEEDED TO CREATE THE OPERATIONAL SYSTEM.

IMPLEMENTATION PHASE

- THE IMPLEMENTATION PHASE CONSISTS OF
  - CODING (I.E. TRADITIONAL PROGRAMMING)
  - UNIT TESTING
  - SOFTWARE INTEGRATION AND TEST
  - SYSTEM TESTING

- WHEN OTHERS TALK OF SOFTWARE DEVELOPMENT THEY NORMALLY MEAN IMPLEMENTATION
  - CURRENTLY WE SPEND A LOT OF TIME THRASHING IN THIS PHASE

- IF ANALYSIS AND DESIGN ARE DONE PROPERLY, IMPLEMENTATION IS A WELL DEFINED EASY PROCESS

INSTRUCTOR NOTES

EACH OF THESE TECHNIQUES CAN HELP CREATE CLEAR, UNDERSTANDABLE PROGRAMS.

IMPLEMENTATION SCOPE

- THE IMPLEMENTATION PHASE CONCERNS ITSELF WITH ISSUES LIKE ...
  - STEP-WISE PROGRAM DECOMPOSITION
  - PROGRAM FAMILIES
  - DATA ABSTRACTIONS AND TYPES
  - DATA STRUCTURES
  - FUNDAMENTAL ALGORITHMS
  - CODING STANDARDS
  - ACCEPTABILITY VS. CORRECTNESS

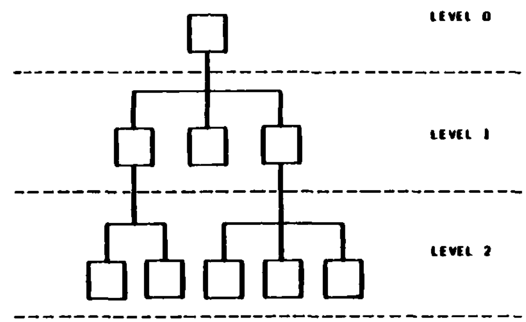
INSTRUCTOR NOTES

HERE, THIS VISUAL TABLE OF CONTENTS DEPICTS THE SUBPROGRAMS CREATED DUE TO STEP-WISE REFINEMENT. WE'VE ALREADY SEEN THIS CONCEPT IN THE DESIGN PHASE.

IMPLEMENTATION ISSUES

(STEP-WISE REFINEMENT)

- STEP-WISE PROGRAM DECOMPOSITION (REFINEMENT) STATED THAT PROGRAMS SHOULD BE WRITTEN IN LEVELS; THE HIGHEST LEVEL CALLING ONE OR MORE SUBPROGRAMS AT THE NEXT LOWEST LEVEL:



- EARLY PROGRAMMING WAS CONTENT TO JUST WRITE A CALL STATEMENT: THAT ALONE WAS CONSIDERED THINKING IN STEP-WISE REFINEMENT TERMS.

- TODAY, STEP-WISE REFINEMENT IS USED PRIMARILY IN THE DESIGN PHASE, WHERE THE TECHNIQUE SHAPES THE SYSTEM'S ARCHITECTURE.

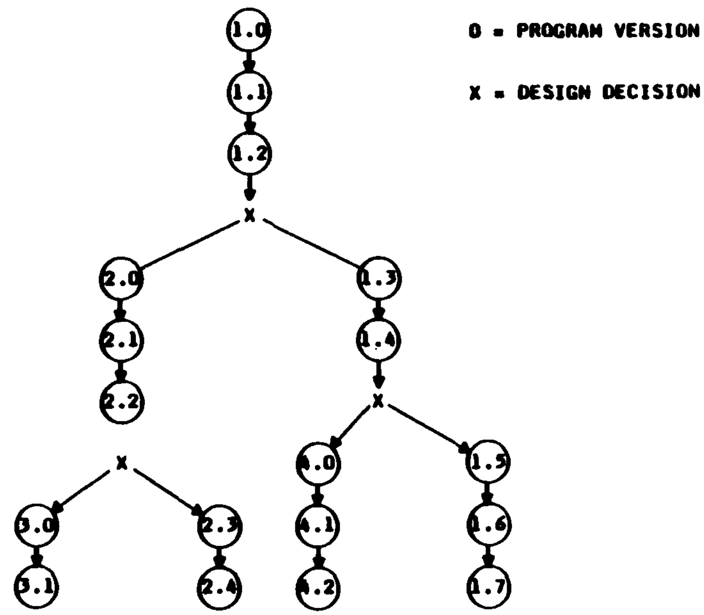
INSTRUCTOR NOTES

HERE, THE NOTATION 1.1 MEANS VERSION 1 PROGRAM 1. THE NOTATION "2" MEANS THIS IS A NEW PROGRAM, SPAWNED FROM ITS ANCESTOR (PROGRAM 1) BY THE CHANGING OF A DESIGN DECISION.

- PROGRAM FAMILIES WORK AT THE MODULE LEVEL. IT DESCRIBES AMOUNT AND TYPES OF IMPACTS WHICH ARE ON THE STRUCTURE OF THE PROGRAM.
- STRUCTURED PROGRAMMING AND PROGRAM FAMILIES ARE NEITHER EQUIVALENT OR CONTRADICTORY. BOTH USE STEP-WISE REFINEMENT TO ENCOURAGE ONE TO MAKE DESIGN DECISIONS EARLY ON.

IMPLEMENTATION ISSUES
(PROGRAM FAMILIES)

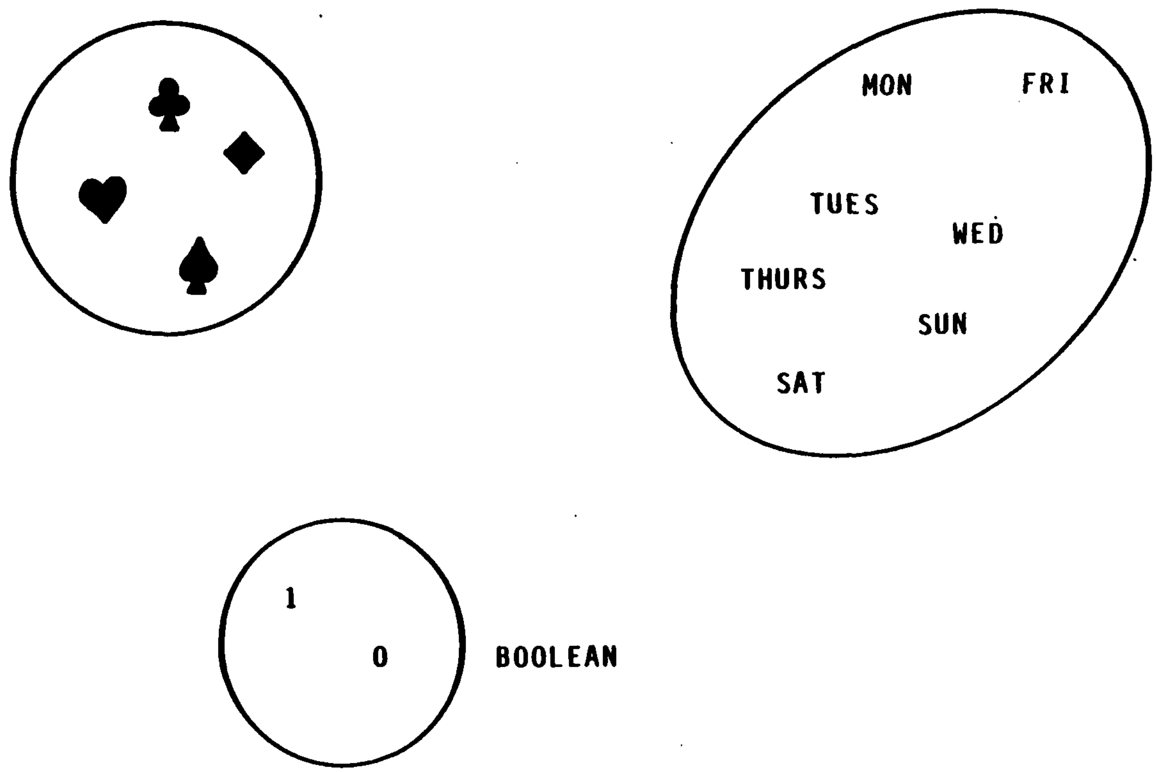
- PROGRAM FAMILIES WAS A CONCEPT THAT STATED THAT A PROGRAM EVOLVES INTO A SET OF RELATED PROGRAMS:



- VERY FEW HAVE WORRIED ABOUT PROGRAM FAMILIES. BUT TODAY'S CONCERN ABOUT THE EVOLUTION OF INDIVIDUAL MODULES AS THEY SIMULTANEOUSLY PARTICIPATE IN SEVERAL SOFTWARE PRODUCTS BRINGS THIS ISSUE AGAIN INTO CENTER STAGE.

INSTRUCTOR NOTES

A PICTURE OF THESE TYPES COULD BE:



IMPLEMENTATION ISSUES

(DATA ABSTRACTIONS AND TYPES)

- DATA ABSTRACTIONS AND TYPES CATEGORIZE PROGRAM VARIABLES AND THEIR VALUES INTO COLLECTIONS HAVING STRONGLY-RELATED CHARACTERISTICS:

```
TYPE SUIT    = (CLUB, DIAMOND, HEART, SPADE);

TYPE DAY     = (MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY,
                SATURDAY, SUNDAY);

TYPE BOOLEAN = (TRUE, FALSE);
```

INSTRUCTOR NOTES

DATA STRUCTURES IMPLY RULES ON HOW TO ACCESS THE DATA THEY CONTAIN.

IMPLEMENTATION ISSUES
(DATA STRUCTURES)

- <u>DATA STRUCTURES</u> CONCERN THEMSELVES WITH ORGANIZING DATA IN WAYS THAT COMPUTERS CAN PROCESS. FOR EXAMPLE ...
  - SCALARS - SINGLE ITEMS (ATOMS) HAVING ONLY A SINGLE VALUE AT ANY GIVEN TIME
  - RECORDS - A LINEAR COLLECTION OF SCALARS OF <u>DIFFERENT</u> TYPE
  - ARRAYS - A LINEAR COLLECTION OF SCALARS OF <u>THE SAME</u> TYPE
  - STACK - A DYNAMIC ARRAY WHERE ELEMENTS ARE TAKEN OFF OF, AND PUT ON AT, THE SAME END
  - QUEUE - A DYNAMIC ARRAY WHERE ELEMENTS ARE PUT ON AT ONE END AND TAKEN OFF THE OTHER END

INSTRUCTOR NOTES

SEE THE KNUTH SERIES FOR MORE EXAMPLES.

IMPLEMENTATION ISSUES

(FUNDAMENTAL ALGORITHMS)

- FUNDAMENTAL ALGORITHMS CONCERN THEMSELVES WITH STANDARD WAYS OF CALCULATING NUMBERS AND MANIPULATING DATA STRUCTURES. SOME OF THEM ARE ...

  - FIBONACCI SEQUENCE GENERATION
  - FACTORIAL COMPUTATION
  - MATRIX MULTIPLICATION
  - LINKED-LIST ALLOCATION
  - TREE MANIPULATION
  - MONITORS
  - INSERTION SORT
  - QUICK SORT

INSTRUCTOR NOTES

STANDARDS SUCH AS THESE ARE PART OF OUR OVERALL MILITARY STANDARDS. THESE ARE TAKEN FROM MIL-STD-SDS. IN SOME CASES THE STANDARD IMPOSES STRICT RULES. IN OTHER CASES THE STANDARD PROVIDES GUIDELINES OR REQUIRES THAT PROJECT SPECIFIC RULES BE DEVELOPED AND ENFORCED.

CODING STANDARDS

- ADDRESS LANGUAGE (HIGH LEVEL VS. ASSEMBLER) TO BE USED
- MAY LIMIT
  - FLOW OF CONTROL CONSTRUCTS
  - NUMBER OF STATEMENTS PER MODULE
  - CODE MODIFICATION
  - SHARING OF VARIABLES, TEMPORARY STORAGE
  - ENTRY AND EXIT POINTS
- PROVIDE COMMENTING GUIDELINES
- ENFORCE NAMING COVENTIONS
- ADRESS CODE READABILITY
  - INDENTING
  - SPACING

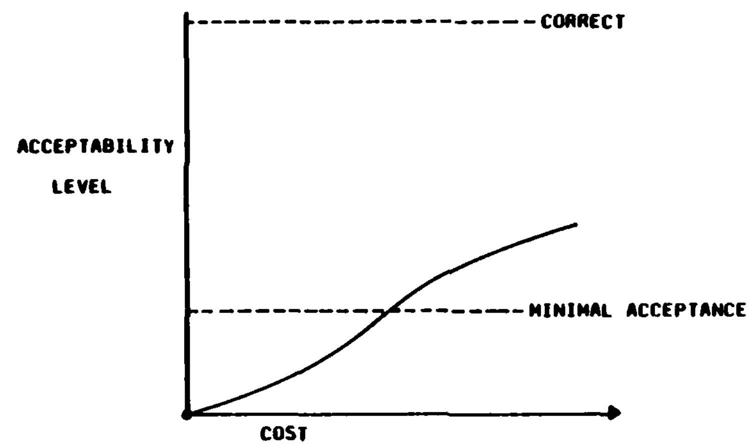
INSTRUCTOR NOTES

IT COSTS A GREAT DEAL TO GET MORE THAN A MINIMAL LEVEL OF ACCEPTABILITY.

ATTAINING CORRECTNESS IS IMPOSSIBLE THESE DAYS.

ACCEPTABILITY VS. CORRECTNESS

- THE ISSUE OF <u>ACCEPTABILITY</u> STATED THAT, SINCE <u>CORRECTNESS</u> COULD NOT BE ATTAINED OF SOFTWARE, TESTING UP TO A LEVEL OF ACCEPTABILITY WAS DEEMED AN ETHICAL PROGRAMMING PRACTICE. THE SAME IS TRUE TODAY.



- THIS IS WHY TESTING, V&V AND QUALITY ASSURANCE HAVE EVOLVED INTO DISCIPLINES OF THEIR OWN -- TO MEET THE NEED OF VALIDATING PROGRAMS TO THE HIGHEST DEGREE OF ACCEPTABILITY POSSIBLE.

INSTRUCTOR NOTES

THEME: TO RELATE THE EARLY 70'S CONCEPTS OF STRUCTURED PROGRAMMING TO MODERN REQUIREMENTS AND Ada.

PURPOSE: TO PROVIDE A LIMITED VIEW INTO STRUCTURED PROGRAMMING.

REFERENCE: WIRTH, N. "PROGRAM DEVELOPMENT BY STEPWISE REFINEMENT" CACM VOL. 14, NO. 4; DECEMBER 1971

DAHL, O., DIJKSTRA, E., HOARE, C. "STRUCTURED PROGRAMMING" ACADEMIC PRESS, LONDON; 1972

SECTION 13

STRUCTURED PROGRAMMING

INSTRUCTOR NOTES

- GIVE SOME MOTIVATION BY DESCRIBING IN GENERAL WHAT STRUCTURED PROGRAMMING IS ABOUT AND WHY IT'S USEFUL. A FEW DIFFERENT DEFINITIONS MAY BE HELPFUL. FOR EXAMPLE,

- EMPHASIZE THAT STRUCTURED PROGRAMMING IS A METHODOLOGY. USING UNIQUE CONTROL STRUCTURES (AND NO GO TO'S) AND NESTING CODE DO NOT BY THEMSELVES CONSTITUTE STRUCTURED PROGRAMMING. ONE ALSO NEEDS AN ORGANIZATIONAL METHOD TO COLLECTIVELY USE THESE TECHNIQUES BY EVERYONE ON A PROJECT, SUCH AS IBM'S PROGRAMMING TEAMS.

- IT IS LANGUAGE INDEPENDENT, BUT CERTAIN LANGUAGES HELP MORE THAN OTHERS, E.G. Ada. STEP-WISE REFINEMENT IS THE KEY.

- IT IS NOT A RELIGION (OF NOT USING GO TO'S). IT IS MEANT TO ENHANCE READABILITY, RELIABILITY, PROGRAMMER EFFICIENCY, ETC. THE ABSENCE OR RARE OCCURRENCE OF GO TO'S IS NO MORE THAN A SYMPTOM OF STRUCTURED PROGRAMMING.

STRUCTURED PROGRAMMING

MOTIVATION

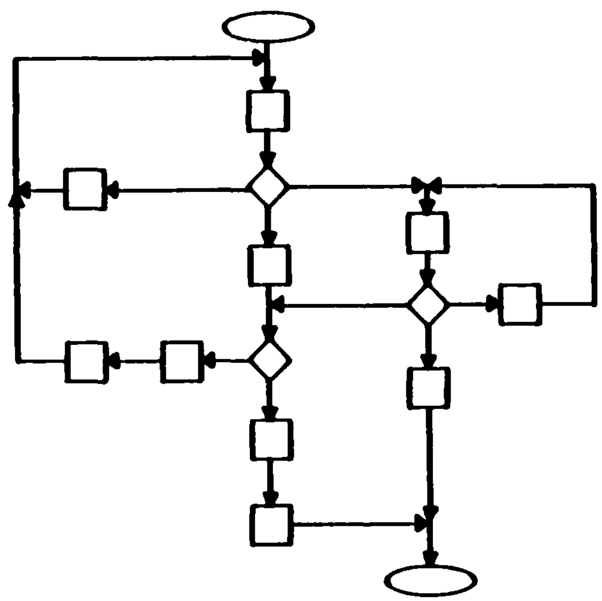
- PROGRAMMING IS A MODELING ACTIVITY AND MODELS MUST BE REVIEWED BY PEOPLE ...
  - TO VERIFY CORRECTNESS OF THE APPROACH
  - TO FIND ERRORS
  - TO SHARE TECHNIQUES
- PROGRAMS MUST BE DESIGNED AND IMPLEMENTED TO BE <u>READ</u> AND <u>UNDERSTOOD</u> BY <u>PEOPLE</u>, NOT JUST COMPUTERS.
- STRUCTURED PROGRAMMING IS A COLLECTION OF TECHNIQUES WHICH EVOLVED TO ANSWER CONCERNS ABOUT THE PROGRAMS WHICH PEOPLE WRITE.

INSTRUCTOR NOTES

- SOFTWARE IS ALWAYS READ MORE OFTEN THAN IT IS WRITTEN.
- STRUCTURED PROGRAMMING KEEPS THINGS SIMPLE, THUS ENHANCING THE LIKELIHOOD THAT THINGS WILL BE CORRECT.
- THIS DIAGRAM REQUIRES A GREAT AMOUNT OF WORK TO DETERMINE WHAT CONDITION HOLDS AT A SPECIFIC POINT. MOREOVER, IT'S NOT CLEAR WHERE THAT POINT EXISTS.

MOTIVATION

- EARLY PROGRAMS WERE JUST WRITTEN WITH NO THOUGHT OF HUMAN CONCERNS. A TYPICAL FLOW DIAGRAM OF SUCH A PROGRAM LOOKED LIKE SPAGHETTI ...



- STRUCTURED PROGRAMMING PROVIDES RULES/GUIDELINES FOR "STRUCTURING" PROGRAMS.

INSTRUCTOR NOTES

ALL THE PREVIOUS METHODOLOGIES HAVE AS THEIR ROOTS STRUCTURED PROGRAMMING. THE FOUNDER IS EDSGER DIJKSTRA. ALTHOUGH IT STARTED OUT AS A REACTION AGAINST "GO TO" TYPE PROGRAMMING, IT NOW HAS NO STANDARD DEFINITION.

STRUCTURED PROGRAMMING

- A DEFINITION ...

"STRUCTURED PROGRAMMING IS THE ART OF ORGANIZING COMPLEXITY, MASTERING MULTITUDES, AND ITS RESULTING BASTARD CHAOS AS EFFECTIVELY AS POSSIBLE."

<u>DIJKSTRA</u>

INSTRUCTOR NOTES

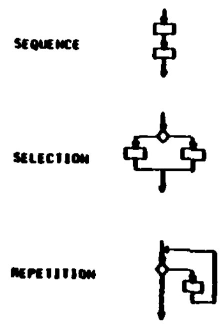
ANY PROGRAM CAN BE WRITTEN USING 3 BASIC CONTROL STRUCTURES.

THE PAPER WAS "FLOW DIAGRAMS, TURING MACHINES, AND LANGUAGES WITH ONLY TWO FORMATION RULES," CACM May 1966.

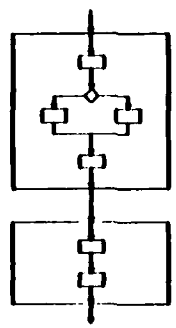
CONTROL STRUCTURING
RULES/GUIDELINES

- BOHM AND JACCOPINNI PROVED THAT <u>ANY</u> LOGIC FLOW COULD BE REDUCED TO THREE SIMPLE RULES ...

- THE THREE BASIC STRUCTURING RULES THEY STATED WERE:



- AND ANY OF THESE RULES CAN BE NESTED HIERARCHICALLY ...



INSTRUCTOR NOTES

WHILE THE THREE FLOW OF CONTROL CONSTRUCTS OF BOHM AND JACOBINE ARE SUFFICIENT, IT HAS BEEN DETERMINED THAT A LIMITED NUMBER OF ADDITIONAL CONSTRUCTS AID UNDERSTANDABILITY. THESE EXAMPLES SHOW WHAT SDS ALLOWS AND WHAT Ada IMPLEMENTS DIRECTLY. NOTE THAT Ada CAN ALSO IMPLEMENT THE Do_Until CONSTRUCT USING A For_Loop WITH AN EXIT STATEMENT AT THE END OF THE LOOP. ALSO NOTE THAT Ada ALLOWS ADDITIONAL CONTROL CONSTRUCTS (I.E., NESTED EXIT STATEMENTS AND EXIT STATEMENTS AT ARBITRARY PLACES IN LOOPS) THAT ARE NOT ALLOWED BY SDS.

CONTROL CONSTRUCTS

MIL-STD-SDS	Ada
• SEQUENCE	• SEQUENCE
• If_Then_Else	• If_Then_Else
• Do_While	• For_Loop
• Do_Until	• While_Loop
• Case	• Case

INSTRUCTOR NOTES

MOST PEOPLE TODAY AGREE THAT STRUCTURED PROGRAMMING IS A SET OF TOOLS AND TECHNIQUES TO SIMPLIFY PROGRAMS AND THEREFORE INCREASE RELIABILITY.

SUMMARY

- THE GOAL OF STRUCTURED PROGRAMMING HAS ALWAYS BEEN TO DEVELOP A STANDARD SET OF RULES FOR STRUCTURING DATA AND ALGORITHMS INTO PROGRAMS.

INSTRUCTOR NOTES

THEME: SOFTWARE TESTING IS MORE THAN DEBUGGING CODE.

PURPOSE: TO PROVIDE AN OVERVIEW OF THE MAJOR STRATEGIES INVOLVED IN SOFTWARE TESTING.

REFERENCE: MILLER, E., HOWDEN, W. "TUTORIAL: SOFTWARE TESTING AND VALIDATION TECHNIQUES" IEEE, EHO 138-8; 1978

SECTION 14

TESTING APPROACHES

INSTRUCTOR NOTES

IF THE TEST RESULT IS CONSISTENT WITH THE EXPECTED RESULT, THE COMPONENT IS DEEMED CORRECT <u>IN THE LIMITED</u> CONTEXT OF THE TEST.

FOR COMPLEX PROGRAMS, THE LIFE CYCLE MAINTENANCE ASPECTS ALSO BECOME IMPORTANT IN CHOOSING THE PROPER TESTING STRATEGY. AREAS THAT CHANGE OFTEN MAY USE A DIFFERENT TYPE OF TESTING THAN AN AREA WHICH DOESN'T.

TESTING

- A TESTED PROGRAM IS ONE WHICH YOU HAVE NOT YET FOUND THE CONDITIONS THAT MAKE IT FAIL.
- TESTING METHODS MUST BE ESTABLISHED ACCORDING TO EACH PROJECT.
  - TYPE OF SYSTEM DIFFERS
  - SIZE OF PROGRAMS VARY
- PRIMARY CATEGORIES OF TESTING
  - UNIT LEVEL
  - INTEGRATION

INSTRUCTOR NOTES

UNIT TEST REQUIRES THE TESTING OF THE LOWEST UNIT OF SOFTWARE INDEPENDENTLY OF OTHER PROGRAMS WHICH INTERACT WITH IT.

UNIT TESTING

- UNIT TESTING IS THE COMPLETE VERIFICATION OF AN INDIVIDUAL MODULE

- UNIT TESTING REQUIRES US TO CONSIDER
  - TESTING APPROACHES
  - TESTING PRINCIPLES
  - TEST CASE DESIGN
  - TESTING STRATEGY

INSTRUCTOR NOTES

BLACK-BOX TESTING USES THE SPECIFICATION TO DEVELOP TEST CASES AND IS MOST APPROPRIATE FOR SYSTEM TESTING.

WHITE-BOX TESTING USES DESIGN INFORMATION TO DEVELOP TEST CASES AND IS MOST APPROPRIATE FOR COMPONENT TESTING.

TESTING APPROACHES

- BLACK-BOX TESTING
  - DOES NOT USE DESIGN KNOWLEDGE
  - REQUIRES <u>EXTERNAL</u> SPECIFICATION
  - SPECIFY INPUTS/OBSERVE OUTPUTS
- WHITE BOX TESTING
  - BASED ON INTERNAL DESIGN LOGIC
  - REQUIRES UNDERSTANDING (DOCUMENTATION) OF DESIGN
  - EMPHASIZES PATH/BRANCH COVERAGE

INSTRUCTOR NOTES

INTEGRATION TESTING HELPS ASSURE THAT THE MODULES OR UNITS WHEN PUT TOGETHER DO INDEED WORK AS A SYSTEM.

INTEGRATION TESTING

- INTEGRATION TESTING IS THE "COMPLETE" VERIFICATION OF THE SET OF MODULES THAT MAKE UP THE SYSTEM
  - BUILDS ON PREVIOUSLY UNIT TESTED MODULES
- INTEGRATION TESTING REQUIRES US TO CONSIDER
  - TESTING APPROACHES
  - INTEGRATION STRATEGIES

INSTRUCTOR NOTES

AT THIS POINT, THE IMPORTANCE OF BOTH UNIT TESTING AND THE INTEGRATION STRATEGY COMES INTO PLAY.

INTEGRATION STRATEGIES

- THE <u>WAY</u> MODULES ARE COMBINED DURING INCREMENTAL TESTING IS CALLED <u>INTEGRATION</u>.

- INTEGRATION TAKES ONE OF TWO FORMS ...

  - TOP DOWN

  - BOTTOM UP

INSTRUCTOR NOTES

THIS STRATEGY IS VERY NATURAL TO A SYSTEMS DEVELOPMENT EFFORT.

TOP-DOWN STRATEGY

- IMPLEMENT THE TOP MODULE OF THE DESIGN FIRST
- SIMULATE THE MODULE'S SUBORDINATES BY STUB MODULES
- GRADUALLY WORK DOWN THE DESIGN, REPLACING STUBS BY REAL MODULES AND INTRODUCING NEW STUBS WHERE NECESSARY

TOP-DOWN INTEGRATION IS THE PREFERABLE APPROACH.

TOP-DOWN INTEGRATION ADVANTAGES

- IMPORTANT FEEDBACK IS PROVIDED TO THE USER IN THE BEST POSSIBLE WAY.
- THE USER MAY TAKE DELIVERY OF SEVERAL SKELETON VERSIONS OF THE SYSTEM, WHICH WILL SMOOTH HIS TRANSITION FROM OLD SYSTEM TO NEW.
- THE PROJECT SHOULD BE IN BETTER POLITICAL SHAPE IF IT FALLS BEHIND SCHEDULE.
- MAJOR INTERFACES ARE TESTED EARLY AND OFTEN.
- IMPLEMENTORS GET A BOOST FROM SEEING SOMETHING WORKING.
- CODING AND TESTING CAN BEGIN BEFORE DESIGN IS FINISHED.

INSTRUCTOR NOTES

DRIVERS ARE USUALLY HARDER TO WRITE THAN STUBS.

BOTTOM-UP INTEGRATION STRATEGY

- BOTTOM-UP INTEGRATION IS, IN GENERAL, THE INVERSE OF TOP-DOWN INTEGRATION ...
- IMPLEMENT A MODULE AT THE BOTTOM OF THE DESIGN.
- SIMULATE THE MODULE'S SUPERORDINATES BY A DRIVER.
- GRADUALLY WORK UP THE DESIGN REPLACING DRIVERS BY REAL MODULES.
- DRIVERS (ALIAS TEST HARNESSES) ARE STAND-INS FOR NOT-YET-WRITTEN SUPERORDINATE MODULES.

INSTRUCTOR NOTES

DRIVERS CAN FORCE A MODULE TO OPERATE UNDER UNUSUAL OR ILLEGAL COMBINATIONS OF INPUTS. A MODULE ONLY OPERATES ON "EXPECTED" DATA. YOU NEED TO KNOW MORE ABOUT THE BEHAVIOR OF A MODULE WHEN ITS BEEN TESTED BOTTOM UP.

BOTTOM-UP INTEGRATION

PLUSES AND MINUSES

+ STARTS OFF WITH GOOD PARALLEL DEVELOPMENT.

- PARALLEL DEVELOPMENT GETS HARDER TO MANAGE AS TEAMS REACH THE TOP OF THE DESIGN.

+ TESTS PHYSICAL I/O INTERFACES EARLY.

- TEST MAJOR INTERNAL INTERFACES LATE.

+ USEFUL FOR SOME CRITICAL MODULES.

- CODING AND TESTING CANNOT BEGIN BEFORE DESIGN FINISHES.

- SKELETON SYSTEMS DIFFICULT TO PRODUCE.

INSTRUCTOR NOTES

SUMMARY

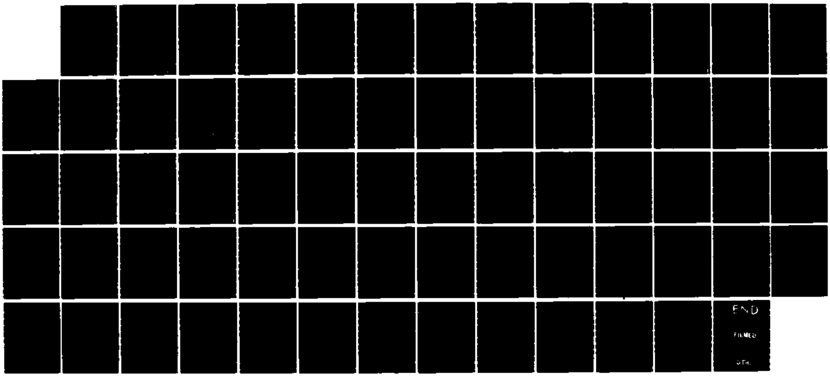
TOP-DOWN VS. BOTTOM-UP

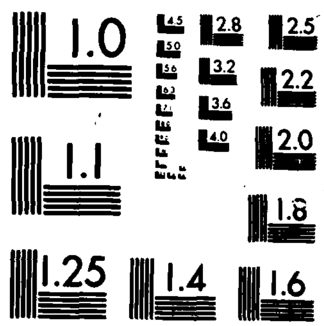
TOP-DOWN	BOTTOM-UP
• BEST FOR TESTING MAJOR INTERFACES EARLY	• BEST FOR TESTING PHYSICAL I/O EARLY
• ASSUMES LOW LEVELS IN DESIGN WILL WORK AS PLANNED	• ASSUMES LOW LEVELS WILL INTEGRATE AS PLANNED
• CONTAINS NO SURPRISES	• VERIFIES LOW-LEVEL ASSUMPTIONS

- IN THE REAL WORLD, A SINGLE STRATEGY WILL NOT WORK. A PROJECT NEEDS THE RIGHT MIXTURE OF BOTH.

AD-A165 123 ADA (TRADEMARK) TRAINING CURRICULUM: SOFTWARE ENGINEERING FOR MANAGERS M101 TEACHER'S GUIDE(U) SOFTECH INC WALTHAM MA 1986 DAAB07-83-C-K506 4/4

UNCLASSIFIED F/G 5/9 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

INSTRUCTOR NOTES

SECTION 15

SOFTWARE MANAGEMENT

INSTRUCTOR NOTES

TOPICS TO BE COVERED INCLUDE

(1) SOFTWARE PLANNING AND TRACKING TECHNIQUES AND TOOLS,

(2) SOFTWARE COST ESTIMATION - BASED PRIMARILY ON BARRY BOEHM'S WORK AND HIS COCOMO MODEL,

(3) QUALITY MANAGEMENT INCLUDING VALIDATION AND VERIFICATION TECHNIQUES, AND

(4) CONFIGURATON MANAGEMENT - WHAT IT IS AND HOW TO GO ABOUT IT.

SOFTWARE MANAGEMENT

- MUST MANAGE THE PROCESS (PROJECT) AND THE PRODUCT (DELIVERABLES)

- ASPECTS TO CONSIDER
  - PLANNING AND TRACKING
  - COST ESTIMATION
  - QUALITY MANAGEMENT
  - CONFIGURATION MANAGEMENT

INSTRUCTOR NOTES

THE NEED FOR PLANNING IS OBVIOUS, BUT IS OFTEN OVERLOOKED. PLANNING SHOULD BE DOCUMENTED. MIL-STD-2167 (SDS) CONTAINS DIDS FOR A SOFTWARE CONFIGURATION MANAGEMENT PLAN, SOFTWARE QUALITY EVALUATION PLAN, SOFTWARE STANDARDS AND PROCEDURES MANUAL, COMPUTER RESOURCE INTEGRITY SUPPORT DOCUMENT, AND MOST IMPORTANTLY THE SOFTWARE DEVELOPMENT PLAN ALL OF WHICH ADDRESS THIS PLANNING ACTIVITY. WE WILL COVER THE GENERAL REQUIREMENTS OF THESE DOCUMENTS, NOT THE SPECIFIC INFORMATION OR FORMATS.

SOFTWARE PLANNING AND TRACKING

- PLANNING IS THE MOST BASIC FUNCTION OF MANAGEMENT
  - DECIDES "WHAT TO DO,"
    "HOW TO DO IT,"
    "WHEN TO DO IT" AND
    "WHO IS TO DO IT"

- TRACKING REPORTS "HOW IT IS GOING"

- PLANNING INVOLVES
  - SETTING OBJECTIVES
  - BREAKING THE WORK INTO TASKS
  - ESTABLISHING SCHEDULES AND BUDGETS
  - ALLOCATING RESOURCES
  - SETTING STANDARDS
  - SELECTING FUTURE COURSES OF ACTION

INSTRUCTOR NOTES

NOTE THE EMPHASIS ON BOTH THE PROJECT (I.E., MONEY, SCHEDULE, STAFFING, ETC.) AND ON THE PRODUCT (I.E., ATTRIBUTES OF THE SPECIFICATION, DESIGN, AND CODE).

WE NEED TO ESTABLISH QUANTITATIVE ESTIMATES IN BOTH AREAS AND HAVE WAYS OF TRACKING AND COMPARING OUR ACTUALS. ALSO NEED CONTINGENCY PLANS WHEN ESTIMATES AND ACTUALS DIVERGE.

SOFTWARE PLANNING AND TRACKING (Continued)

- TWO PRIMARY ASPECTS TO BE TRACKED
  - THE SOFTWARE PRODUCT
  - THE SOFTWARE PROJECT

- PURPOSE
  - MEASURE AND EVALUATE PROGRESS
  - EARLY IDENTIFICATION OF PROBLEMS WHILE THERE IS STILL TIME TO TAKE CORRECTIVE ACTION

- KEYS TO SUCCESSFUL TRACKING
  - BASE TRACKING ON DELIVERABLES (I.E., SOMETHING THAT CAN BE MEASURED AS COMPLETE OR NOT)
  - ESTABLISH MILESTONES FOR ALL DELIVERABLES
  - HAVE AN ESTIMATE TO COMPARE ACTUALS TO (RELIES ON PLANNING)

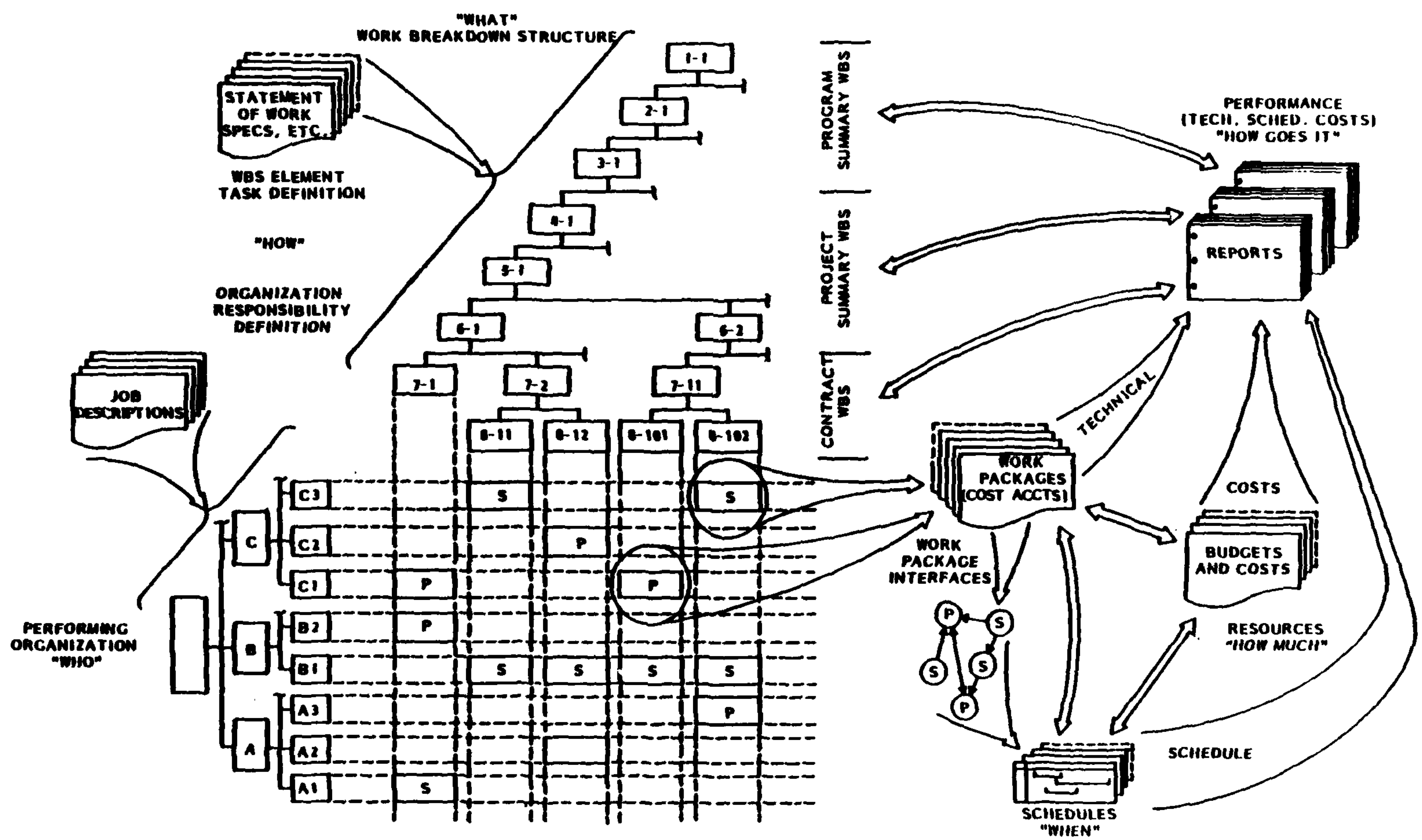
INSTRUCTOR NOTES

PLANNING ACTIVITIES HAVE TO WORK WITHIN TYPICAL MATRIX ORGANIZATIONS. THE PROJECT STRUCTURE REFLECTED BY THE WBS IS SHOWN ON TOP. THE PERFORMING ORGANIZATION IS SHOWN ON THE LEFT. THE INTERSECTIONS DEFINE THE SPECIFIC WORK PACKAGES (PEOPLE AND TASKS) THAT CONTRIBUTE TO THE DELIVERED PRODUCT.

SCHEDULES DEFINE "WHEN" THESE WORK PACKAGES GET PERFORMED.

SOFTWARE PLANNING AND TRACKING

(STRUCTURING TECHNIQUES)



P = Principal Responsibility
S Support Responsibility

INSTRUCTOR NOTES

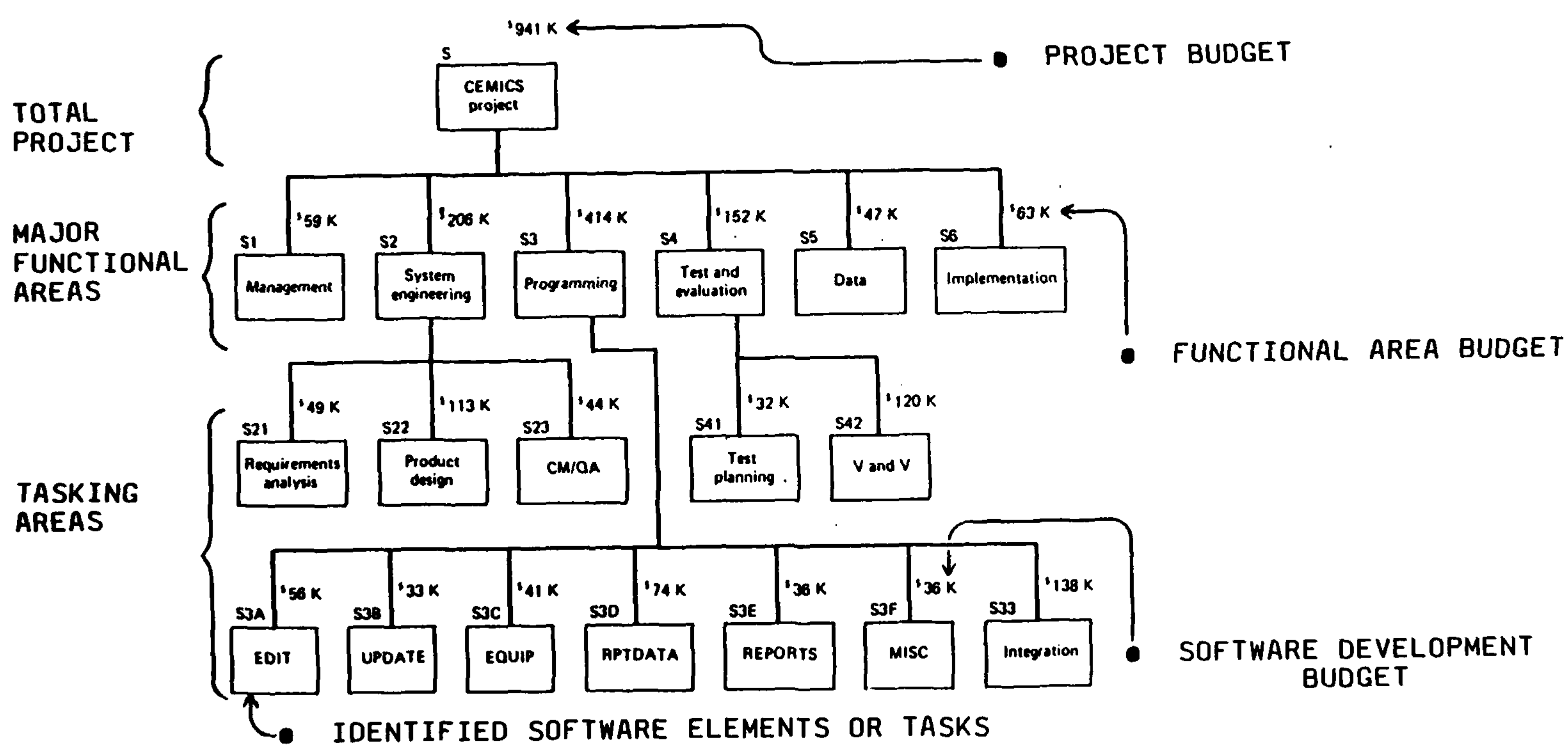
THE WBS REPRESENTS THE TASK WE MUST PERFORM IN A HIERARCHICALLY STRUCTURED FORM. THIS EXAMPLE SHOWS THREE LEVELS, BUT ON LARGE PROJECTS MORE LEVELS ARE OFTEN USED.

MANY OTHER ASPECTS OF THE PROJECT ARE RELATED TO OR MAY BE DRIVEN BY THE WBS. WE MAY BE REQUIRED TO PRODUCE A PLAN FOR EACH FIRST LEVEL ITEM, OR TO TRACK STATUS AT SOME PARTICULAR LEVEL (I.E., THE THIRD).

COMMON PROBLEMS WITH WBSs ARE OMISSIONS (IT WON'T GET DONE) AND REDUNDANCY (TWO SEPARATE GROUPS MAY BE DOING IT).

THE WBS

- CHARACTERISTICS OF THE WORK BREAKDOWN STRUCTURE (WBS)
  - A TREE STRUCTURE THAT REPRESENTS ALL TASKS AND SERVICES NEEDED IN DEVELOPING SOFTWARE
  - PRESENTS TASKING AT SUMMARY AS WELL AS DETAIL LEVEL



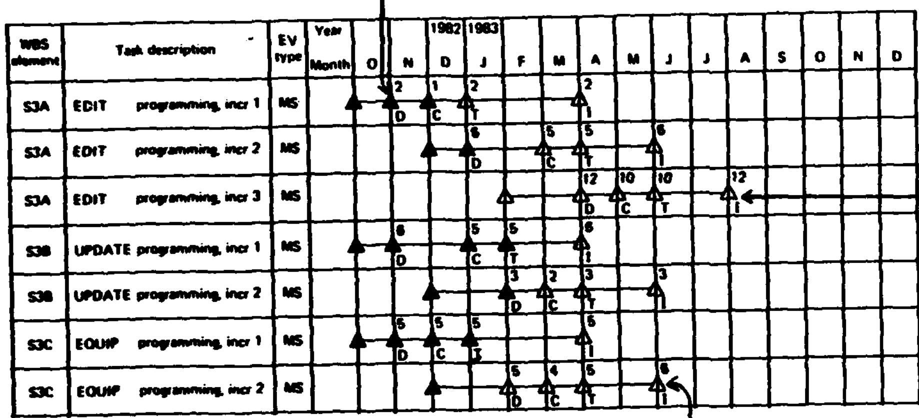
INSTRUCTOR NOTES

GNATT CHARTS SHOW OVERALL PROJECT SCHEDULES, WITHOUT ANY OF THE INTERDEPENDENCIES SHOWN ON A PERT CHART.

THESE CHARTS SHOW WHAT HAS HAPPENED (TRIANGLES THAT ARE FILLED IN) AND WHAT IS STILL SCHEDULED (OPEN TRIANGLES). THESE TRIANGLES USUALLY REPRESENT MILESTONES OF ONE FORM OR ANOTHER.

SCHEDULES

(USING GNATT CHARTS)



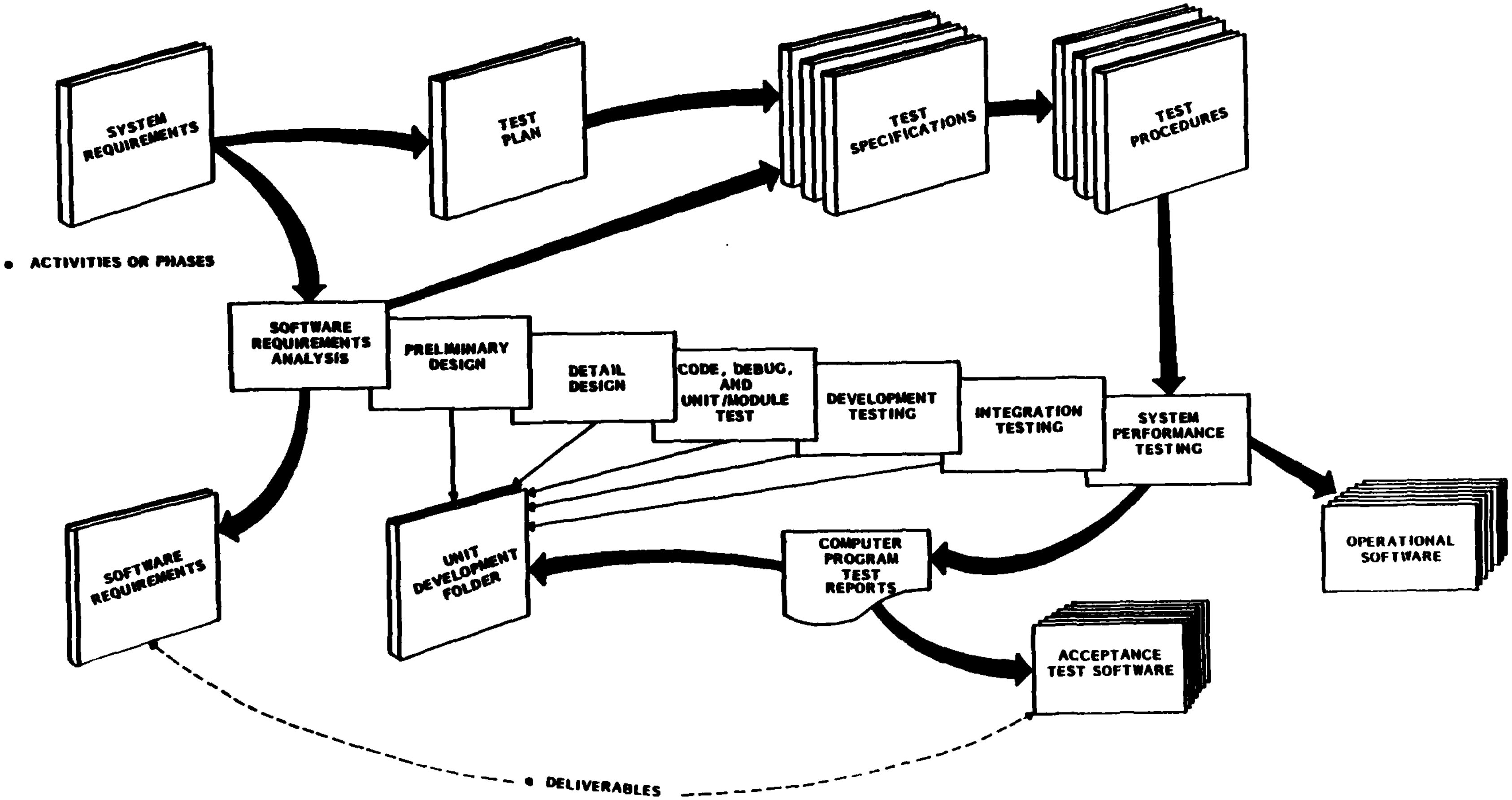
- A COMPLETED MILESTONE
- ACTIVITY TYPE
  (D = DESIGN
  C = CODE, ETC.)
- A MILESTONE <u>NOT</u> COMPLETED

- PROVIDES A VIEW OF OVERALL SCHEDULE AND COMPLETION STATUS

INSTRUCTOR NOTES

NOTE THAT THIS IS SHOWING GENERIC SOFTWARE DELIVERABLES - IT IS NOT SPECIFICALLY DERIVED FROM MIL-STD-2167 (SDS) OR ANY OTHER STANDARD.

SOFTWARE DEVELOPMENT DELIVERABLES



INSTRUCTOR NOTES

UDFs ARE SORT OF A "GRASS ROOTS" CONCEPT THAT ARE KNOWN BY SEVERAL OTHER NAMES (SOFTWARE DEVELOPMENT FILE, MODULE DEVELOPMENT FILE, MODULE DEVELOPMENT FOLDER, AND MANY OTHERS). IT IS REALLY A CONFIGURATION MANAGEMENT SCHEME FOR INTERMEDIATE SOFTWARE DEVELOPMENT WORK PRODUCTS.

IN SOME ORGANIZATIONS THE UDF IS LEVELED OFF FROM DEVELOPER TO INTEGRATION AND TEST WHEN THE "UNIT" HAS PASSED ITS UNIT TEST.

MUCH UDF INFORMATION IS A CANDIDATE FOR KEEPING ON-LINE, SO THAT IT CAN BE PROCESSED BY VARIOUS TOOLS AND CAN BE USED TO GENERATE STATUS REPORTS.

UNIT DEVELOPMENT FOLDER

- A FORMALIZED PROGRAMMERS NOTEBOOK
  - PROVIDES A UNIFORM AND VISIBLE COLLECTION POINT FOR ALL UNIT DOCUMENTATION AND CODE
  - PROVIDES MANAGEMENT VISIBILITY INTO THE DEVELOPMENT PROCESS

- UDF CONTAINS FOR A UNIT
  - REQUIREMENTS
  - DESIGNS
  - TEST PLAN
  - CODE
  - TEST RESULTS
  - PROBLEM REPORTS
  - NOTES
  - REVIEWERS' COMMENTS

- IN ADDITION A UDF SUMMARIZES THE STATUS OF A UNIT

INSTRUCTOR NOTES

MUCH OF THE CRITICISM OF SOFTWARE FOR ALWAYS EXCEEDING ITS COST ESTIMATES IS BASED ON OUR POOR ABILITY TO PRODUCE AND JUSTIFY RELIABLE ESTIMATES. THIS IS AN AREA OF SOFTWARE ENGINEERING THAT NEEDS SUBSTANTIAL IMPROVEMENT.

TO BE ABLE TO PRODUCE ESTIMATES WE NEED

(1) COST MODELS - (I.E., THE RELATIONSHIP BETWEEN LINES OF CODE AND COST, BETWEEN LANGUAGE AND COST, BETWEEN EXPERIENCE AND COST, ETC.)

(2) COST MONITORING PROCEDURES - BECAUSE FUTURE PROJECT COSTS ARE MOST CLOSELY RELATED TO COSTS-TO-DATE.

(3) COST HISTORY MAINTENANCE - THIS IS HOW WE BUILD OUR MODELS AND OUR PROJECTION ABILITIES - WITHOUT GOOD COST DATA FROM SIMILAR PROJECTS WE HAVE ALMOST NO ABILITY TO MAKE RELIABLE PROJECTIONS.

SOFTWARE COST ESTIMATION

- MOTIVATION
  - EVERY PRODUCT MUST BE <u>ENGINEERED TO COST</u>
  - COSTS MUST BE <u>PREDICTED</u>

- SOFTWARE COST ESTIMATION IS THE PROCESS OF DETERMINING THE FULL COST OF THE DEVELOPMENT OF SOFTWARE

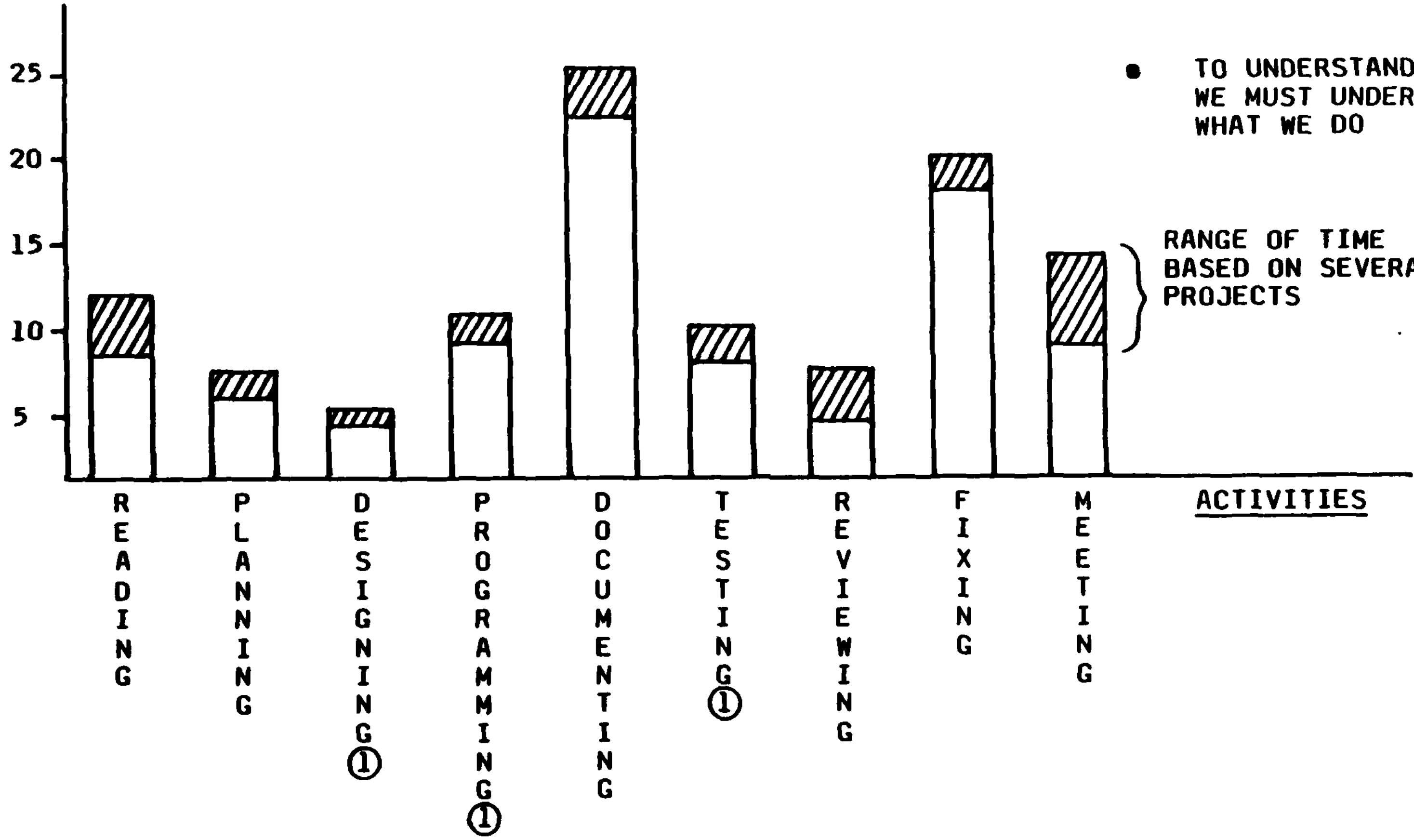
- SOFTWARE COST ESTIMATION REQUIRES:
  - COST MODELING TECHNIQUES
  - COST MONITORING PROCEDURES
  - COST HISTORY MAINTENANCE

INSTRUCTOR NOTES

THIS GRAPHIC POINTS OUT HOW WE REALLY DON'T HAVE MUCH INSIGHT INTO WHAT SOFTWARE ENGINEERS DO WITH THEIR TIME. WE TALK ABOUT DESIGN, CODE AND TEST AS THEIR PRINCIPLE ACTIVITIES, BUT IN REALITY THIS IS ONLY 25 - 30%.

NOTE THAT JUST DELETING AN EXPENSIVE ACTIVITY (DOCUMENTING) DOESN'T REDUCE COSTS - IT JUST MOVES IT ELSEWHERE (FIXING).

ACTIVITIES OF A SOFTWARE PROJECT



- TO UNDERSTAND COSTS WE MUST UNDERSTAND WHAT WE DO

① TRADITIONAL ACTIVITIES INCLUDED IN COST ESTIMATES

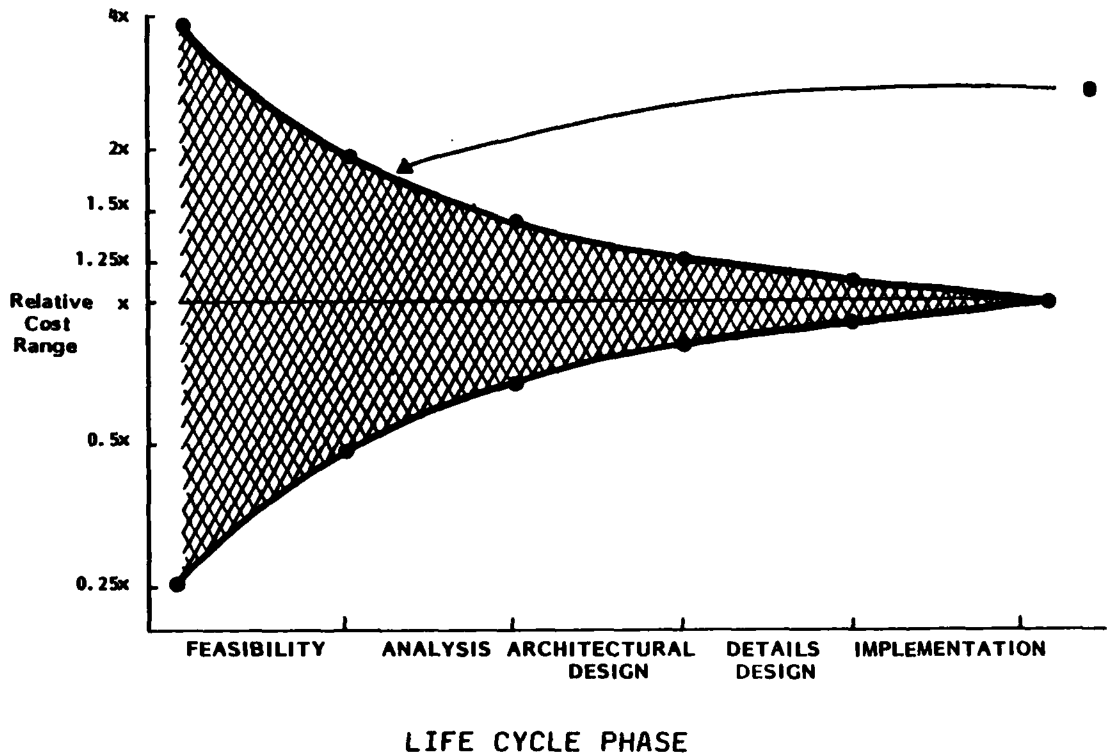
- TRADITIONAL ESTIMATES TYPICALLY ONLY ACCOUNT FOR 25% TIME ASSOCIATED WITH A SOFTWARE PROJECT

INSTRUCTOR NOTES

THIS DIAGRAM DRAMATICALLY SHOWS THE RELATIVE ACCURACY OF COST ESTIMATE DEPENDING ON THE PHASE OF A PROJECT. DURING EARLY FEASIBILITY PHASES, IT IS NOT UNUSUAL FOR ESTIMATES TO BE HIGH OR LOW BY A FACTOR OF 4 (I.E., ESTIMATING ANYWHERE FROM $4 MILLION TO $250,000 FOR WHAT IS ULTIMATELY A $1 MILLION DOLLAR JOB). EVEN IN THE MIDDLE OF THE ANALYSIS PHASE, AT THE POINT WE WOULD EXPECT A FAIRLY GOOD REQUIREMENTS SPECIFICATION TO BE AVAILABLE WE COULD STILL BE OFF BY A FACTOR OF 2.

SOFTWARE COST ESTIMATION

EXPECTED ACCURACY VS PHASE



- UNCERTAINCIES DUE TO
  - LACK OF UNDERSTANDING OF REQUIREMENTS
  - POOR ESTIMATION SKILLS OF DESIGNERS
  - A THOUSAND OTHER FACTORS

SOURCE: BOEHM, SOFTWARE ENGINEERING ECONOMICS, 1982

INSTRUCTOR NOTES

COCOMO - <u>CO</u>NSTRUCTIVE <u>CO</u>ST <u>MO</u>DEL WAS DEVELOPED BY BARRY BOEHM AND IS THE SUBJECT OF HIS BOOK ON SOFTWARE ENGINEERING ECONOMICS. COCOMO IS AVAILABLE IN SEVERAL LEVELS OF COMPLEXITY (BASIC, INTERMEDIATE, DETAILED). THE MODEL IS BASED ON EXTENSIVE EMPIRICAL DATA.

LIKE ALL SUCH MODELS CERTAIN PARAMETERS PLAY A VERY SIGNIFICANT ROLE IN THE ACCURACY OF RESULTS - <u>THE MOST IMPORTANT PARAMETER IS THE ESTIMATED NUMBER OF LINES OF CODE</u> - NEXT MOST IMPORTANT ARE CHARACTERISTICS OF THE PRODUCT BEING DEVELOPED (I.E., ITS COMPLEXITY, SIZE AND SPEED CONSTRAINTS, ETC.) AND THE QUALITIES OF THE STAFF AND CHARACTERISTICS PROJECT SUCH AS SCHEDULE AND FACILITIES.

A SOFTWARE COST ESTIMATION TECHNIQUE - COCOMO

- COCOMO IS BASED ON EMPIRICAL DATA
  - OVER 63 MAJOR PROJECTS
  - ACCURATELY MODELS SOFTWARE COSTS TO 20% OF ACTUAL COSTS 80% OF THE TIME

- SOFTWARE COSTS DETERMINATION DEPENDS ON
  - ESTIMATING SOURCE LINES OF CODE
  - CHARACTERIZING THE SOFTWARE PRODUCT
  - CHARACTERIZING THE SOFTWARE PROJECT
  - USING THE COCOMO EQUATIONS

INSTRUCTOR NOTES

THESE CHARACTERIZATIONS ARE THE MOST SIGNIFICANT INFLUENCE OF THE BASIC MODEL. SOFTWARE COSTS ARE ALWAYS CHARACTERIZED BEING EXPONENTIALLY RELATED TO THE SIZE OF THE PRODUCT. WHICH CATEGORY OUR PRODUCT FALLS INTO AFFECTS A CONSTANT MULTIPLIER AND THE ACTUAL EXPONENT. (YOU CAN WORK OUT A COUPLE OF EXAMPLES TO SHOW HOW THE EXPONENT CAUSES SIGNIFICANT NON-LINEAR GROWTH AS SIZES GET LARGE.)

$2.8\ (40)^{1.2}$ = 234 MM FOR 40,000 LINE EMBEDDED SYSTEM

$2.8\ (80)^{1.2}$ = 538 MM FOR 80,000 LINE EMBEDDED SYSTEM

$2.8\ (400)^{1.2}$ = 3712 MM for 400,000 LINE EMBEDDED SYSTEM

$3.2\ (400)^{1.05}$ = 1727 MM FOR 400,000 LINE ORGANIC SYSTEM

$3.2\ (40)^{1.05}$ = 154 MM FOR 40,000 LINE ORGANIC SYSTEM

THESE FIGURES SHOW THAT EMBEDDED SYSTEMS ARE MUCH MORE EXPONENTIALLY AFFECTED BY SIZE THAN ORGANIC SYSTEMS.

CHARACTERIZING THE SOFTWARE PRODUCT

- CATEGORIZE THE SOFTWARE
  - ORGANIC MODE
    - LIGHTLY CONSTRAINED
    - FAMILIAR SITUATION
    - EXAMPLE - DATA REDUCTION SOFTWARE
  - EMBEDDED MODE
    - TIGHTLY CONSTRAINED
    - UNFAMILIAR SITUATION
    - EXAMPLE - AIRCRAFT COLLISION AVOIDANCE SOFTWARE
  - SEMI-DETACHED MODE
    - BETWEEN ORGANIC AND EMBEDDED
    - EXAMPLE - TRAINER SOFTWARE

- NOMINAL COST IN MAN-MONTHS
  - ORGANIC : $MM_{NOM} = 3.2\ (KDSI)^{1.05}$
  - SEMI-DETACHED : $MM_{NOM} = 3.0\ (KDSI)^{1.12}$
  - EMBEDDED : $MM_{NOM} = 2.8\ (KDSI)^{1.20}$

INSTRUCTOR NOTES

THESE ARE ADDITIONAL MULTIPLICATIVE FACTORS THAT ARE USED IN COMING UP WITH THE FINAL MAN-MONTH ESTIMATE. NOTE THAT THE MOST IMPORTANT FACTORS ARE THE ONES WITH THE WIDEST RANGE NOT NECESSARILY THE LARGEST OR SMALLEST ABSOLUTE NUMBER (I.E., ANALYST CAPABILITY, PRODUCT COMPLEXITY).

WHETHER THE RANGE IS INCREASING OR DECREASING ISN'T IMPORTANT - JUST A NOTATIONAL CONVENIENCE - MUST SEE DETAILED DESCRIPTION TO UNDERSTAND.

CHARACTERIZING THE SOFTWARE PROJECT

- COST DRIVERS IN SOFTWARE RESULT IN EFFORT MULTIPLIERS

- REQUIRED SOFTWARE RELIABILITY	[0.75 .. 1.40]
- DATABASE SIZE	[0.94 .. 1.16]
- PRODUCT COMPLEXITY	[0.70 .. 1.65]
- EXECUTION TIME CONSTRAINT	[1.00 .. 1.66]
- MAIN STORAGE CONSTRAINT	[1.00 .. 1.56]
- VIRTUAL MACHINE VOLATILITY	[0.87 .. 1.30]
- COMPUTER TURNAROUND TIME	[0.87 .. 1.15]
- ANALYST CAPABILITY	[1.46 .. 0.71]
- APPLICATION EXPERIENCE	[1.29 .. 0.82]
- PROGRAMMING CAPABILITY	[1.42 .. 0.70]
- VIRTUAL MACHINE EXPERIENCE	[1.21 .. 0.90]
- PROGRAMMING LANGUAGE EXPERIENCE	[1.14 .. 0.95]
- USE OF MODERN METHODS	[1.24 .. 0.82]
- USE OF TOOLS	[1.24 .. 0.83]
- REQUIRED DEVELOPMENT SCHEDULE	[1.23 .. 1.10]

- EFFORT MULTIPLIER
  - π = (PRODUCT OF EACH OF THE FACTORS ABOVE AS SELECTED FOR YOUR PROJECT).

INSTRUCTOR NOTES

THESE ARE THE FINAL STEPS OF ACTUALLY COMING UP WITH A PRICE TAG. IT IS THE PRODUCT OF ALL THE CHARACTERIZATIONS FROM THE PREVIOUS SLIDE.

STRESS THE IMPORTANCE OF THE ASSUMPTIONS IN PARTICULAR INSTRUCTION COUNT AND THE DIFFICULTY IN GETTING A GOOD ESTIMATE FOR KDSI.

ALSO POINT OUT THE CUMULATIVE EFFECT OF ALL THE OTHER CHARACTERIZATIONS.

USING THE COCOMO EQUATIONS

- ESTIMATING EFFORT IN MAN-MONTHS

$$MM = MM_{NOM} \cdot \pi$$

- ESTIMATING COST IN DOLLARS

$$COST = MM \cdot COST_{MM}$$

WHERE $COST_{MM} \approx$ AVERAGE COST PER MAN-MONTH FOR THE PROJECT

- REMEMBER ESTIMATES ARE ONLY AS ACCURATE AS YOUR ASSUMPTIONS
  - DELIVERED INSTRUCTION COUNT
  - CHARACTERISTICS OF PRODUCT AND PROJECT

INSTRUCTOR NOTES

NOTE THAT WHILE WE USUALLY TALK ABOUT QUALITY IN GENERAL TERMS WE CAN ESTABLISH QUANTITATIVE MEASURES SUCH AS NUMBER OF ERRORS PER 1000 LINES OF CODE OR PERCENT OF TRACEABLE REQUIREMENTS OR COMPLEXITY MEASURES ON SOFTWARE MODULES.

THE LAST POINT IS THE ONE TO BE STRESSED ON THIS SLIDE - QUALITY IS THE RESULT OF PLANNING AND MUST BE INTEGRATED INTO THE ENTIRE DEVELOPMENT PROCESS - QUALITY CAN'T JUST BE TESTED IN AT THE END OF A PROJECT.

SOFTWARE QUALITY MANAGEMENT

- WHAT IS "QUALITY"?
  - AN ESSENTIAL NATURE OR CHARACTERISTIC
  - THE DEGREE OF EXCELLENCE

- WHAT IS "SOFTWARE QUALITY"?
  - DEGREE OF EXCELLENCE EXHIBITED BY A SOFTWARE PRODUCT (WHERE A SOFTWARE PRODUCT CONSISTS OF PROGRAMS AND OTHER RELEVANT DOCUMENTATION NEEDED TO SPECIFY, DESIGN, CODE, TEST, USE AND MAINTAIN THOSE PROGRAMS)

- FOR SOFTWARE QUALITY MANAGEMENT TO BE SUCCESSFUL IT MUST BE TREATED AS AN INTEGRAL PART OF THE DEVELOPMENT PROCESS

INSTRUCTOR NOTES

VERIFICATION AND VALIDATION AND SOFTWARE QUALITY ASSURANCE ARE SOMETIMES USED INTERCHANGEABLY. IN FACT VERIFICATION AND VALIDATION ARE TECHNIQUES FOR ASSURING SOFTWARE QUALITY (THE GOAL).

VEPIFICATION IS A PROCESS PERFORMED AT EACH STEP AND COVERS JUST THAT ONE STEP.

VALIDATION IS AN END-TO-END PROCESS WHICH ATTEMPTS TO MAKE SURE THE COMPLETED SYSTEM MEETS ITS OVERALL OBJECTIVES.

SOFTWARE QUALITY MANAGEMENT IS A PROCESS OF VERIFICATION AND VALIDATION

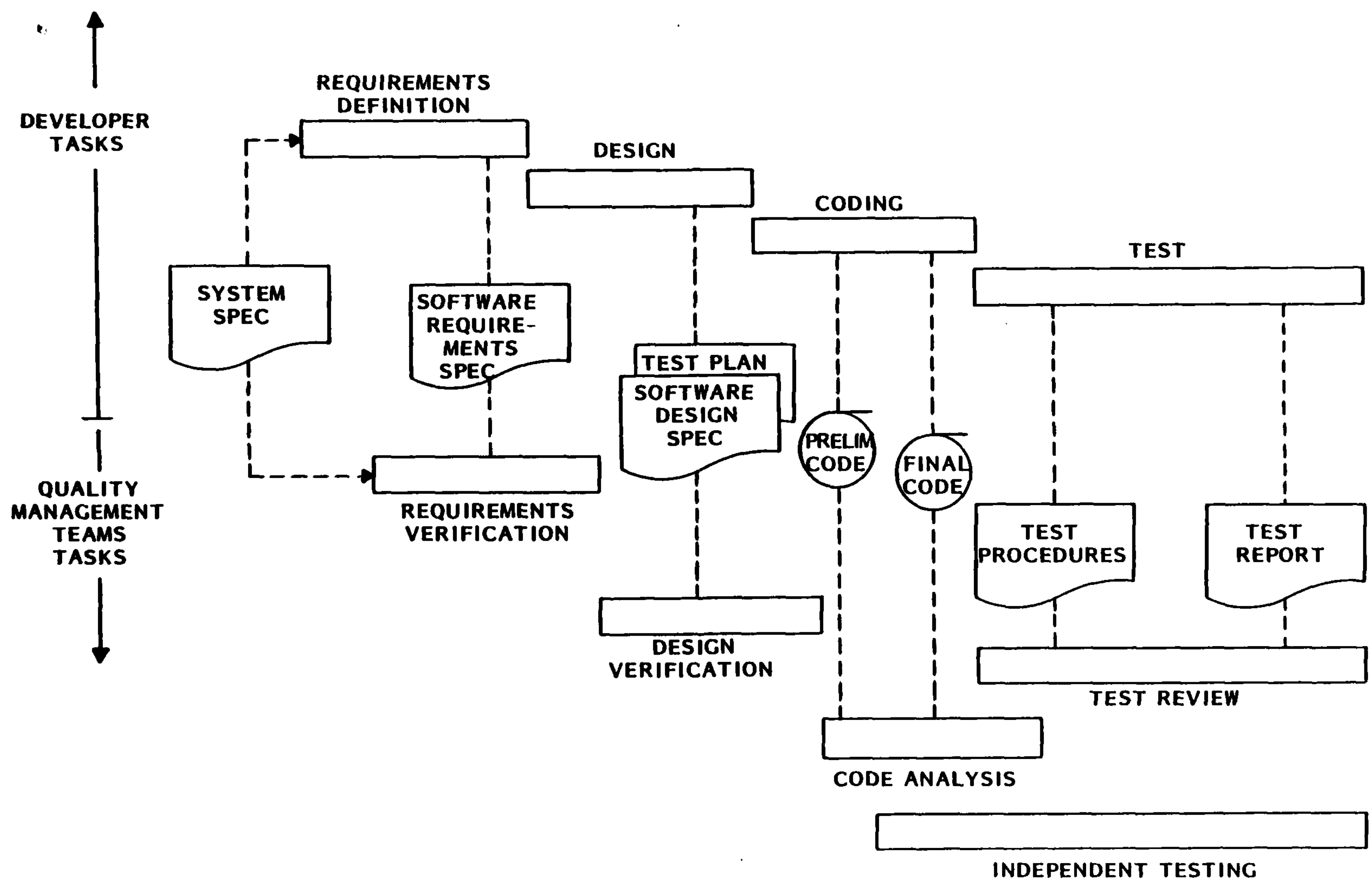
- VERIFICATION - ITERATIVE PROCESS OF DETERMINING WHETHER EACH STEP OF THE DEVELOPMENT PROCESS FULFILLS <u>ALL THE REQUIREMENTS</u> LEVIED BY THE PREVIOUS STEP.
  - IS THE PRODUCT COMPLETE?
  - IS THE PRODUCT CONSISTENT?

- VALIDATION - EVALUATION, INTEGRATION AND TEST ACTIVITIES CARRIED OUT AT THE SYSTEM LEVEL TO ENSURE THE PRODUCT IS MEETING <u>ALL REQUIREMENTS</u>.
  - IS THE RIGHT PROBLEM BEING SOLVED?
  - IS THE SOLUTION ACCURATE?
  - IS THE PRODUCT USABLE?

INSTRUCTOR NOTES

THIS SLIDE SHOWS THE VERIFICATION STEPS (REQUIREMENTS VERIFICATION, DESIGN VERIFICATION, CODE ANALYSIS, AND TEST REVIEW). ALSO SHOWN IS INDEPENDENT TESTING WHICH IS THE VALIDATION TASK. IT WOULD PROBABLY BE APPROPRIATE FOR THE SYSTEM SPECIFICATION TO BE SHOWN AS PROVIDING INPUTS TO THIS INDEPENDENT TESTING TASK.

VERIFICATION AND VALIDATION TASKS



VG 742.1

INSTRUCTOR NOTES

MOST OF THESE TECHNIQUES ARE VERIFICATION TECHNIQUES - THEY ARE APPLIED TO ONE OR MORE STEPS, BUT EACH APPLICATION ONLY COVERS THAT PARTICULAR STEP.

THE FINAL THREE TECHNIQUES ARE PART OF THE VALIDATION TASK.

VERIFICATION AND VALIDATION TECHNIQUES

VERIFICATION AND VALIDATION TECHNIQUE	LIFE CYCLE PHASE				RELATIVE COST
	ANALYSIS	DESIGN	IMPLEMENTATION		
			CODE	TEST	
REQUIREMENTS TRACEABILITY	X	X	X	X	MEDIUM
INTERFACE ANALYSIS	X	X	X	X	MEDIUM
QUALITY ASSESSMENT	X	X	X	X	MEDIUM
SECURITY EVALUATION	X	X	X	X	LOW
INDEPENDENT STRUCTURED ANALYSIS	X	X			HIGH
FAILURE MODES & EFFECTS ANALYSIS	X	X	X	X	MEDIUM
DESIGN WALKTHROUGH	X	X	X	X	LOW
PROTOTYPING		X	X		HIGH
MODULE INDEPENDENCE ANALYSIS		X	X		MEDIUM
CODE INSPECTION			X		LOW
CODE WALKTHROUGH			X		LOW
STATIC CODE ANALYSIS			X		LOW
DYNAMIC CODE ANALYSIS			X	X	HIGH
TEST PLANS/PROCEDURES EVALUATION				X	LOW
INDEPENDENT TESTING				X	MEDIUM
TEST WITNESSING/EVALUATION				X	LOW

INSTRUCTOR NOTES

THERE ARE THREE MAJOR ASPECTS TO CM

(1) IDENTIFYING - NAMING, COUNTING, NUMBERING, RELATIONSHIPS, ETC.

(2) TRACKING - THE CHANGES IN THESE IDENTIFIED ENTITIES OVER TIME

(3) STATUS REPORTING - OF THE IDENTIFICATION AND TRACKED CHANGES

CONFIGURATION MANAGEMENT

- CONFIGURATION MANAGEMENT IS THE APPLICATION OF TECHNICAL AND ADMINISTRATIVE CONTROL TO:

  - IDENTIFY AND DOCUMENT PHYSICAL CHARACTERISTICS OF CONFIGURATION ITEMS (SOFTWARE DELIVERABLES) AND THE RELATIONSHIPS AMONG THESE ITEMS

  - TRACK THOSE CHARACTERISTICS AND RELATIONSHIPS

  - RECORD AND REPORT CHANGE PROCESSING AND STATUS

INSTRUCTOR NOTES

ON SMALL SIMPLE SYSTEMS, WE CAN PROBABLY GET AWAY WITH ONLY CONCERNING OURSELVES WITH THE IDENTIFICATON OF CODE - NOT NECESSARILY DISTINGUISHING BETWEEN SOURCE, OBJECT, AND EXECUTABLE IMAGE. AS SYSTEMS GET LARGER AND MORE COMPLEX (AND IN PARTICULAR AS A SYSTEM IS DEPLOYED IN DIFFERENT CONFIGURATIONS) WE MUST ALSO IDENTIFY THE COMPONENTS OF THE DOCUMENTATION, THE TOOLS (COMPILERS) USED TO PROCESS THE CODE AND DATA, AND DISTINGUISH BETWEEN THE DIFFERENT FORMS OF CODE.

CONFIGURATION ITEMS AND IDENTIFICATION

- ANY ITEM WHICH IS A PRODUCT OF A SOFTWARE DEVELOPMENT ACTIVITY
  - SYSTEM DOCUMENTATION
  - SOURCE CODE
  - OBJECT CODE
  - EXECUTABLE IMAGES
  - PROGRAMMING SUPPORT TOOLS

- IT MUST BE POSSIBLE TO UNIQUELY IDENTIFY EVERY ITEM UNDER CONFIGURATION CONTROL

INSTRUCTOR NOTES

THERE ARE TWO ASPECTS TO THIS POINT

(1) CONTROLLING WHO IS ALLOWED TO CHANGE WHAT (THIS IS MORE IMPORTANT ON AUTOMATED SYSTEMS THAN FOR MANUAL ONES), AND

(2) KEEPING A RECORD OF THE ALLOWABLE CHANGES.

THESE CONTROLS APPLY TO THE FORMAL PART OF THE SYSTEM - THE OFFICIAL BASELINE - PRIVATE CHANGES THAT DON'T AFFECT THE SHARED OR CONTROL BASELINE - ARE USUALLY ALLOWED AND ARE NOT GENERALLY TRACKED.

CHANGE CONTROL AND TRACKING

- PROGRAMMERS SHOULD BE ALLOWED ACCESS ONLY TO THOSE CONFIGURATION ITEMS THEY NEED
- IT SHOULD BE POSSIBLE TO MAKE PRIVATE EXPERIMENTAL CHANGES THAT DO NOT AFFECT OTHER PROGRAMMERS
- MECHANISMS TO CHANGE AND DOCUMENT SOFTWARE MUST BE PROVIDED
- CHANGE DOCUMENTATION INCLUDES AT LEAST:
  - WHO MADE THE CHANGE
  - WHY THE CHANGE WAS MADE
  - DESCRIPTION OF THE CHANGE
  - DATE OF THE CHANGE

INSTRUCTOR NOTES

STATUS ACCOUNTING IS PRIMARILY CONCERNED WITH PRODUCING MANAGEMENT REPORTS - BUT IT SHOULD ALSO BE ABLE TO RESPOND TO SPECIFIC QUERIES OR ALLOW THE TRACING AND/OR RECONSTRUCTION OF ANY ITEM AT ANY TIME.

STATUS ACCOUNTING

- MECHANISM FOR MAINTAINING A RECORD OF HOW A CONFIGURATION ITEM HAS EVOLVED
- PROVIDES STATUS OF THE CONFIGURATION ITEM AT ANY TIME
- PROVIDES REPORTING CAPABILITIES
  - PROJECT STATUS VISIBILITY TO MANAGEMENT
  - CHANGE CONSISTENCY FOR CM PERSONNEL
  - REQUIRED CHANGE INFORMATION FOR PROGRAMMERS

INSTRUCTOR NOTES

AGAIN THIS IS A SUMMARY OF THE MOST VISIBLE, OFTEN USED INFORMATION PROVIDED BY A CM SYSTEM. A SYSTEM SUCH AS ALS AUTOMATICALLY RECORDS AND MAINTAINS ALL OF THIS INFORMATION.

THIS TYPE OF INFORMATION ALLOWS FOR THE AUTOMATIC RECREATION OF ANY PREVIOUSLY BUILT SYSTEM.

CONFIGURATION MANAGEMENT

- A CM SYSTEM MAINTAINS:
  - CREATION DATES, CREATOR, AND REASON FOR CREATION
  - DERIVED-FROM/DERIVES-TO RELATIONS
  - CONFIGURATION AND STATUS INFORMATION FOR BUILT SYSTEMS
  - VERSIONS OF CONFIGURATION ITEMS (A HISTORY)
  - VARIATIONS OF CONFIGURATION ITEMS (AN ALTERNATE IMPLEMENTATION)

INSTRUCTOR NOTES

IT IS IMPORTANT TO POINT OUT THAT EVEN WHEN AUTOMATED TOOLS ARE AVAILABLE (SUCH AS IN ALS) THAT AN OVERALL SCHEME OR CM STRATEGY (OR A POLICY) STILL NEEDS TO BE DEVELOPED AND ENFORCED. TOOLS USUALLY JUST PROVIDE CM CAPABILITIES, EVEN IF THEY ARE INTEGRATED. EITHER THESE TOOLS NEED TO BE CONFIGURED OR CONSTRAINED BY A MASTER TOOL OR ADDITIONAL MANAGEMENT PROCEDURES NEED TO BE PUT IN PLACE.

SUMMARY

- CONFIGURATION MANAGEMENT
  - CONTROLS THE DETAILS OF CHANGE
  - REQUIRES AN OVERALL SCHEME
  - REQUIRES INTEGRATED TOOLS
  - TRACKS VERSIONS

INSTRUCTOR NOTES

IV. SUMMARY/SOFTWARE ENGINEERING AND Ada - ALLOW 1/4 HOUR

SUMMARIZE THE DAY AND SHOW THE RELATIONSHIP OF SOFTWARE ENGINEERING WITH Ada

PART IV

SOFTWARE ENGINEERING AND Ada

INSTRUCTOR NOTES

SECTION 16

RELATIONSHIP OF SOFTWARE ENGINEERING TO Ada

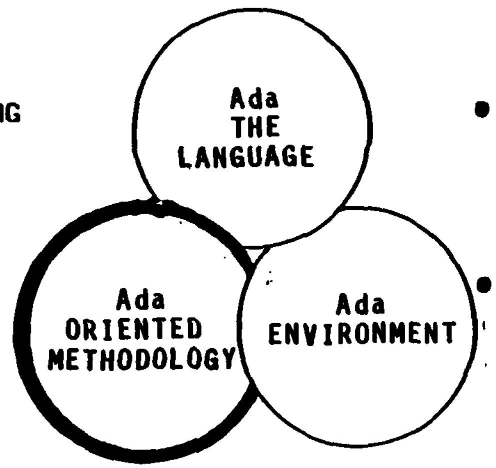
INSTRUCTOR NOTES

THIS SLIDE SHOWS HOW THE DoD VIEW THE TOTAL Ada EFFORT.

RELATIONSHIP OF "Ada THE LANGUAGE" AND METHODOLOGIES

- ORIENTED TO OUR APPLICATIONS
- MANAGES THE COMPLEXITY ASSOCIATED WITH SOFTWARE SYSTEMS

- REDUCE COST OF DEVELOPING SYSTEMS
- INCREASE PORTABILITY OF SOFTWARE AND DEVELOPERS



- IMPROVE PRODUCTIVITY
- INCREASE SOFTWARE RELIABILITY, MAINTAINABILITY

THREE ASPECTS WHICH ADDRESS THE "SOFTWARE PROBLEM".

INSTRUCTOR NOTES

THESE ARE THE STATED GOALS OF THE METHODMAN DOCUMENT, AND TO SOME EXTENT THE STARS PROGRAM (AS OF EARLY 1985).

Ada ORIENTED METHODOLOGY

- GOAL IS TO ENCOURAGE THE USE OF AND SUPPLEMENT THE DEVELOPMENT OF THE SET OF METHODS USED IN THE PROCESS OF MANAGING, DEVELOPING, AND MAINTAINING SOFTWARE SYSTEMS
- MUST ADDRESS FULL LIFE CYCLE
- KEY ASSUMPTION - "INCREASED EFFORT IN EARLIER STAGES OF DEVELOPMENT WILL BE REFLECTED IN REDUCED COST FOR TESTING AND MAINTENANCE"
- FOCUS COMES FROM
  - METHODMAN
  - STARS (SOFTWARE TECHNOLOGY FOR ADAPTABLE AND RELIABLE SYSTEMS)

INSTRUCTOR NOTES

DETAILS OF METHODMAN GOALS.

"METHODMAN" GOALS

GENERAL GOALS:

- BALANCE BETWEEN SIMPLICITY AND COMPLEXITY
- CONTROL OF COMPLEXITY
- RIGOR
- APPLY TO ANY PROBLEM DOMAIN

TECHNICAL GOALS:

- CRITERIA GIVEN FOR ALL TECHNICAL ASPECTS
- FORMALIZATION OF SPECIFICATIONS AND DESIGN
- VERIFICATION OF SPECIFICATION AND DESIGN DECISIONS
- PROVIDE AN EXPLICIT MODEL OF THE REAL WORLD
- OVERALL OPTIMIZATION OF LOGICAL/PHYSICAL DATA AND PROCESSING ARCHITECTURES
- SUPPORT DESIGN OF CONCURRENT HARDWARE AND SOFTWARE SYSTEMS

PRODUCT MANAGEMENT GOALS:

- PROVIDE QUALITY CONTROL
- PROVIDE VERSION CONTROL
- PROVIDE CONFIGURATION MANAGEMENT
- PROVIDE AN EXPLICIT MODEL OF THE SOFTWARE DEVELOPMENT PROCESS

"A GOOD METHODOLOGY"

AUTOMATION GOALS:

- AUTOMATE LIFE CYCLE PROCESSES THAT ARE CONVENTIONALLY DONE MANUALLY AND REDUNDANTLY
- INTEGRATED FAMILY OF TOOLS
- PROVIDE GRAPHICAL TOOLS

PROJECT MANAGEMENT GOALS:

- IMPROVE THE MANAGEABILITY OF SOFTWARE PRODUCTION
- IMPROVE THE EFFICIENCY OF SOFTWARE PRODUCTION
- IMPROVE THE SE PRACTICES OF PROGRAMMERS IN THE ORGANIZATION
- 100% CENTRALLY VERIFIED CONSISTENCY
- IMPROVE PRODUCTIVITY OVER THE ENTIRE LIFE CYCLE

INSTRUCTOR NOTES

DETAILS OF STARS PROGRAM GOALS.

THE STARS PROGRAM

WILL FOCUS Ada METHODOLOGY DEVELOPMENT

OBJECTIVES

- IMPROVE ENVIRONMENTS
  - IMPROVE PERSONNEL RESOURCE
    - INCREASE LEVEL OF EXPERTISE
    - EXPAND BASE OF EXPERTISE AVAILABLE TO DoD
  - IMPROVE POWER OF TOOLS
    - IMPROVE PROJECT MANAGEMENT TOOLS
    - IMPROVE APPLICATION-INDEPENDENT TECHNICAL TOOLS
    - IMPROVE APPLICATION-SPECIFIC TECHNICAL TOOLS
  - INCREASE USE OF TOOLS
    - IMPROVE BUSINESS PRACTICES
    - IMPROVE TOOL USABILITY
    - INCREASE LEVEL OF TOOL INTEGRATION
    - INCREASE LEVEL OF TOOL AUTOMATION

"... TO IMPROVE PRODUCTIVITY WHILE ACHIEVING GREATER SYSTEM RELIABILITY AND ADAPTABILITY"

Material: Software Engr'g for Managers (M101), Teacher's Guide

We would appreciate your comments on this material and would like you to complete this brief questionaire. The completed questionaire should be forwarded to the address on the back of this page. Thank you in advance for your time and effort.

1. Your name, company or affiliation, address and phone number.

2. Was the material accurate and technically correct?

Yes ☐ No ☐

Comments:

3. Were there any typographical errors?

Yes ☐ No ☐

If yes, on what pages?

4. Was the material organized and presented appropriately for your applications?

Yes ☐ No ☐

Comments:

5. General Comments:

place
stamp
here

COMMANDER
US ARMY MATERIEL COMMAND
ATTN: AMCDE-SB (OGLESBY)
5001 EISENHOWER AVENUE
ALEXANDRIA, VIRGINIA 22233

END

FILMED

DTIC